Management of the Internet and Complex Services

European Sixth Framework Network of Excellence FP6-2004-IST-026854-NoE

Deliverable D9.4 Emerging Paradigms and Technologies for Autonomic Management

The EMANICS Consortium

Caisse des Dépôts et Consignations, CDC, France
Institut National de Recherche en Informatique et Automatique, INRIA, France
University of Twente, UT, The Netherlands
Imperial College, IC, UK
International University Bremen, IUB, Germany
KTH Royal Institute of Technology, KTH, Sweden
Oslo University College, HIO, Norway
Universidat Politecnica de Catalunya, UPC, Spain
University of Federal Armed Forces Munich, UniBwM, Germany
Poznan Supercomputing and Networking Center, PSNC, Poland
University of Zürich, UniZH, Switzerland
Ludwig-Maximilian University Munich, LMU, Germany
University College London, UCL, UK
University of Pitesti, UniP, Romania

For more information on this document or the EMANICS Project, please contact:

Dr. Olivier Festor
Technopole de Nancy-Brabois — Campus scientifique
615, rue de Jardin Botanique — B.P. 101
F—54600 Villers Les Nancy Cedex
France

Phone: +33 383 59 30 66
Fax: +33 383 41 30 79
E-mail: <olivier.festor@loria.fr>

Document Control

Title:	Emerging Paradigms and Technologies for Autonomic Management
Type:	Public
Editors:	George Pavlou, Oscar Fredy Gonzalez Duque, Marinos Charalambides.
E-mail:	g.pavlou@ee.ucl.ac.uk, O.Gonzalez-Duque@surrey.ac.uk, m.charalambides@ee.ucl.ac.uk
Authors:	Arosha K Bandara, Thomas Bocek, Marinos Charalambides, Thibault Cholez, Isabelle Christment, Oscar Fredy Gonzalez Duque, David Hausheer, Fabio Hecht, Iris Hochstatter, Thiery Kramis, Raul Landa, Feng Liu, Emil C Lupu, Christian Morariu, George Pavlou, Dalibor Peric, Peter Racz, Javier Rubio Loyola, Martin Serrano, Joan Serrat, Burkhard Stiller, Han Manh Tran (in alphabetical order)
Doc ID:	D9.4-v0.2.doc

AMENDMENT HISTORY

Version	Date	Author	Description/Comments
V0.1	November 15, 2008	Marinos Charalambides	First version - ToC
V0.2	March 06, 2009	See authors list	Initial draft for internal commenting
V0.3	March 10, 2009	G Pavlou, O. F. Gonzalez Duque, M. Charalambides	Proof reading and editing
V1.0	March 31, 2009	O. F. Gonzalez Duque, M. Charalambides	Final editing.

Table of Contents

(This page is left blank intentionally.)

Executive Summary

Computing and communication systems have benefited from a constant and relentless evolution of hardware and software that has rendered their components sophisticated yet inexpensive. This has allowed for the development of large scale systems composed of vast numbers of devices with different computational and communication capabilities that interact using a wide variety of physical media and communication protocols. Although these systems commonly exhibit desirable characteristics such as flexibility, dynamicity and heterogeneity, they also pose a problem to system administrators – and often to users – due to the intrinsic level of complexity that they bring to the management of IT systems. In this respect, autonomic management has emerged as an appealing solution that unloads the management burden from system administrators and leaves it on the system itself. Thus, from a holistic point of view an autonomic management system can be thought of as a self-governing system equipped to make decisions in accordance to system goals and surrounding conditions in the absence of human administrators. Consequently, management systems need to display a minimum set of properties, known as the self-* properties, to be deemed autonomic: self-awareness, self-configuration, self-optimisation, self-healing and self-protection.

EMANICS Network of Excellence through its Work Package 9 (Autonomic Management) aims at providing systems with the self-* properties to enable autonomic management by making use of technologies, frameworks and approaches developed through research integration and collaboration among WP9 partners. Within this context, for the 2nd Phase of EMANICS four tasks of noticeable importance and interest have been identified by WP9 partners:

- *Task 1: Intra & inter-domain autonomic management of fixed networks*. This task is one of the core activities for WP9. A lot of effort is aimed at addressing inter-domain issues as they pose one of the most challenging aspects of management in general.
- *Task 2: Autonomic management of ubiquitous environments*. In addition to autonomic management principles, this task also includes activity addressing protection and context exploitation.
- *Task 3: Policy based autonomic management*. The focus of this task is policies since they have been identified as a critical paradigm and emerging technology that can potentially facilitate autonomic management.
- *Task 4: Peer-to-peer approaches for autonomic network management*. The emergence of peer-to-peer technologies and their dramatic penetration in today's networks motivate the investigation of such approaches for autonomic management.

Each activity presented in this deliverable (D9.4) addresses various aspects of one or more of the tasks above as they have several overlapping areas being targeted by the WP9 partners' joint research efforts.

Deliverable D9.4, as originally intended, builds on research work performed as part of 1st period of EMANICS Phase 2 which was presented in deliverable D9.3. As a result four out of the six activities presented in this document are a continuation of 1st period activities. Thus, D9.4 marks a vital middle-way milestone within EMANICS Phase 2 and in particular within WP9 context. Not only does it capitalise on previous research efforts, but it also readies WP9 partners for the final push towards sophisticated systems capable of truly realising autonomic management.

1 Introduction

Managing large scale systems composed of vast numbers of heterogeneous devices that communicate with each other by means of a wide variety of physical media and communication protocols is a very complex and expensive task. The problem is exacerbated due to the dynamic and volatile nature of many of today's networks where users, in addition to roaming freely, can typically enter and leave the network without any warning thus making many of the network services intermittent. For instance, it is easy to foresee the daunting task encountered by administrators in charge of networked environments consisting of thousands of sensors and hundreds of mobile users spanning over a building a few tens of floors high. The IT management experts do never seem to be enough to maintain control over fixed networks let alone dynamically changing networks. This is the reason why autonomic management is such an appealing solution to the ever growing complexity of management systems. With autonomic management the burden of managing a system is passed from the administrators to the system itself. This means that the system becomes self-managed or self-governed leaving to the administrators the responsibility of steering the system's overall behaviour through desirable high-level objectives. However, achieving autonomic management requires research work to encompass different areas due to the extensiveness of the autonomic management field. In this light, WP9 has set up six activities – four of which are continuing activities from 1st period of EMANICS Phase 2 – that address different aspects of autonomic management with state of the art approaches and technologies.

In recent years, policy-based management has been proposed as a suitable means for dealing with different aspects of management systems. Polices offer a clean approach to differentiate the functionality of a system from the rules employed to lead the system to a desired state. Such an approach gives administrators the ability to change the state of the system without having to recode its functionality, instead they can steer the system in a non-intrusive manner by giving it a new set of high-level objectives/rules. Additionally, the system can monitor itself and its surroundings so as to adapt its functionality according to the current conditions and the system objectives. This provides autonomic behaviour in the form of a closed feedback loop that can be applied locally to individual system components or hierarchically to sets of cooperating components. Within this area of policy-based management systems two aspects emerge that are of special importance: policy refinement and policy conflict analysis and resolution. Policy refinement is essential to guarantee that high-level, abstract policies (objectives) are correctly and efficiently transformed into the low-level, concrete policies that directly control the system. On the other hand, conflict analysis and resolution provides configuration stability by dealing with conflicting policies that may lead the system to an unpredictable state. In Section 2 we explore these critical aspects of policy based management.

Highly available and resilient networks play a decisive role in today's networked world. Although in these networks planning is of great importance to reduce disruptions, it is relevant to keep in mind than system faults will inevitably occur. Therefore, in order to alleviate possible system faults it is imperative to develop mechanisms to assist administrators in the generation and implementation of effective recovery plans to minimise service disruptions. Section 3 targets intricate fault recovery issues using automated mechanisms based on emerging technologies such as case-based reasoning, peer-to-peer technology and artificial intelligence-based automated planning approaches.

Context information modelling has acquired relevance in next generation networks and their context-aware services. In such networks the enormous amount of context information constantly produced by the environment sources makes its gathering, aggregation and analysis complex. Additionally, the complexity for managing context information grows exponentially with the abundance of services and diversity of service requirements, which places new requirements on information management and control. Section 4, presents a solution that provides a functional concept for context information management in different provider domains addressing questions of modelling, exchange protocols, accounting, inconsistency and resilience on behalf of future autonomic communications. In this respect, the use of ontologies for support and management of services (OSM) facilitates the transition from service agnostic management architectures to service and network resource-awareness by proposing functional components that exhibit autonomic principles.

In real-time traffic analysis one of the key operations is capturing the traffic from a network link. However, the “everything-over-IP” approach adopted in today’s networks has had a direct impact on traffic analysis. Each new service offered in a network imposes a significant increase in the amount of carried traffic, especially for multimedia oriented services. Such traffic increases highly impact the ability of network operators to perform traffic analysis, which they require to understand the type of traffic they carry and be able to plan network management accordingly. Moreover, traffic profiling in a network is particularly important for operators to be able of providing better quality services, plan network upgrades, detect malicious traffic, or charge users depending on the traffic they generate. Section 5 introduces two complementary approaches tailored for high-speed links. The first deals with distributed packet capturing while the second takes care of distributed storage of IP flow records; both approaches are essential in the analysis of real-time network traffic.

The Future Internet Networks (FINs) paradigm implicitly requires that both network and service providers offer and publish their services so that more complex services can be provided. In such scenes, virtual service broker negotiators take care of the service requests and provide support to organise the underlying service providers in order to provide the appropriate composition tasks. This support should be carried out taking into account the plethora of service providers, the services they provide, their interests, their service qualities and other key aspects. In addition, this functionality should be provided freeing the requesting entities from any complex decision-making processes. Thus, a virtual service negotiation broker needs to implement a consistent and distributed service coalition mechanism. Section 6 addresses this problem with a mechanism that provides support for the service coalition formation process for Future Internet virtual service negotiation brokers. The proposed mechanism contemplates several aspects envisaged necessary for FIN-oriented solutions.

Emerging P2P storage networks are able to provide highly-available access to huge amounts of data by leveraging the resources of multiple inexpensive and untrusted storage devices distributed over a wide area. However, this solution presents some problems that are currently unsolved. Incentives for storage provider and consumer must be in place for the network to provide a satisfactory quality of service. Also, high performance bandwidth requirements in P2P storage networks are not always met. A globally unique identifier in a P2P storage network makes changes difficult to be tracked as most P2P storage systems use a hash value generated based on the content. A small change in the content leads to a totally different hash. Moreover, a P2P storage system needs to cope with malicious peers that can degrade performance and reliability. Finally, in a fully distributed P2P storage system, there is no access control and administrative methods in place. Section 7 presents a set of complementary solutions that addresses these unsolved problems in the emerging P2P storage networks.

1.1 Purpose of the Document

The purpose of this document – deliverable D9.4 – is to provide a detailed view of state of the art emerging paradigms and technologies for autonomic management. Deliverable D9.4 the outcome of ongoing research integration and collaboration among partners of EMANICS Network of Excellence Work Package 9 (WP9: Autonomic Management). As intended by the EMANICS partners, work presented in this deliverable enhances and builds on research work developed during 1st period of EMANICS Phase 2 which was presented in deliverable D9.3. Additionally, this document puts in evidence that WP9 has succeeded in building strong collaboration and cooperation research links between partners, which is imperative to achieve the goals set up by the EMANICS project.

This document presents the research results produced by the six activities approved during the second open call period of EMANICS Phase 2. Four activities are the continuation of research carried out during the 1st period of Phase 2, while the other two activities have been set up in other to address different areas of the autonomic management field.

1.2 Document Outline

This document is organised as follows. Section 1 provides a general introduction, and states the purpose and outline of this document. In Section 2 we present extensive work performed in the area of policy-based autonomic management in the context of intra and inter-domain network management. Section 3 examines emerging technologies such as case-based reasoning, peer-to-peer technology and artificial intelligence-based automated planning as means to address intricate fault recovery issues in autonomic systems. In Section 4 we propose a solution that provides a functional concept for context information management in different provider domains following the principles of autonomic management. Section 5 presents two approaches for autonomic real-time traffic analysis that deal with distributed packet capturing and distributed storage of IP flow records. A mechanism that provides support for the service coalition formation process in future Internet networks (FINs) is introduced in Section 6. Section 7 proposes a set of complementary solutions that addresses unsolved problems in the emerging P2P storage networks. Finally, Section 8 summarises and concludes the work presented in this deliverable.

2 Policies for Intra & Inter Domain Autonomic Management (PINIDAM2)

2.1 Introduction

Policy-based approaches to network and systems management are of particular importance because they allow the separation of the rules that govern the systems behaviour from the functionality provided by those systems. This means that it is possible to adapt the behaviour of a system without the need to recode functionality or to stop the system. This provides autonomic behaviour in the form of an efficient closed feed-back loop either locally or hierarchically. In particular, policy based techniques are intimately linked to the principles of Autonomic Management. For instance, self-configuration, self-optimisation and self-protection are key properties of Autonomic Systems and is widely accepted that policy-based management can address their requirements effectively.

Within the framework of Autonomic Systems, policy-based management has also been proposed as a suitable means for managing different aspects of IP networks, including Quality of Service (QoS) and security protection. Yet despite various research projects, standardisation efforts and substantial interest from industry, policy-based management is still not a reality. There are some vendor tools, mostly for virtual private network provisioning, but policy-based management is still far from being widely adopted despite its potential benefits of flexibility and "constrained programmability". One of the reasons behind the reticence to adopt this technology is that it is difficult to analyse policies in order to guarantee configuration stability given that policies may have conflicts leading to unpredictable effects. The work presented here attempts to tackle this problem through two complementary approaches applied to two important fronts in autonomic management systems, i.e. quality of service management for differentiated services networks and firewall configuration management. Both approaches enhance the capabilities and improve the suitability of policy-based frameworks to support intra & inter domain autonomic management. Additionally, our research further develops previous work carried out in PINIDAM phase one and, at the same time, prepares the road for the use of policies in phase 3. We introduce our approach based on the formalisation and reasoning provided by event calculus in Sections 2.2, 2.3 and 2.4. Sections 2.5, 2.6 and 2.7 show how formal argumentation logic and preference reasoning are used to manage firewall configuration.

2.2 Policy Driven Quality of Service Management

QoS management has always been one of the most popular application domains of policies since ISPs can realise their objectives through flexible programmability with respect to offered services and treatment of customer traffic in their network. A small number of policies have been defined for some of the components of the QoS management framework [1]. Here, we provide a comprehensive set of QoS management policies and explain how their enforcement defines the behaviour of the managed modules and associated IP DiffServ managed objects. The majority of these policies are generic enough to apply to other QoS and resource management frameworks, where functions for admission control and BW management are essential. They are categorised into service management and traffic engineering policies and follow the two-level hierarchy of Figure 2-1.

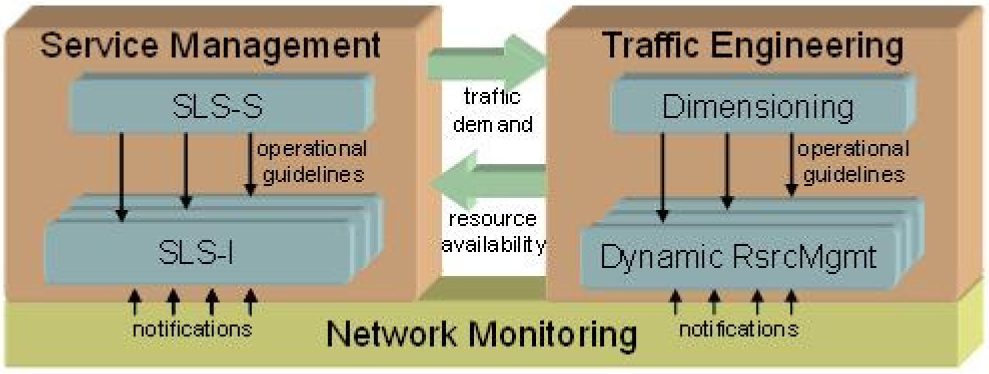


Figure 2-1. QoS Management Framework. Two main sub-systems, Service Management and Traffic Engineering, each implementing a two-level module hierarchy.

2.2.1 Service Management Policies

The service management functionality is realised by the Service Level Specification-Subscription (SLS-S) and Service Level Specification-Invocation (SLS-I) modules that perform static and dynamic admission control respectively. The main objective of subscription logic is to control the number and type of service subscriptions, aiming to avoid overloading the network, whilst at the same time maximising subscribed traffic. To achieve this objective, the SLS-S module employs managed objects that expose a set of methods defining its programmable functionality. These methods guide the evolution of the SLS-S module in its operation through a number of states represented in the state machine diagram of Figure 2-2. The transitions between these states can thus be managed through policies. The SLS-S operation distinguishes two main processes – initialisation and admission control – during which parameters essential for the modules operation are initialised, and the actual decision on the acceptance/rejection of new subscriptions is taken; the relevant policy actions are summarised on Table 2-1.

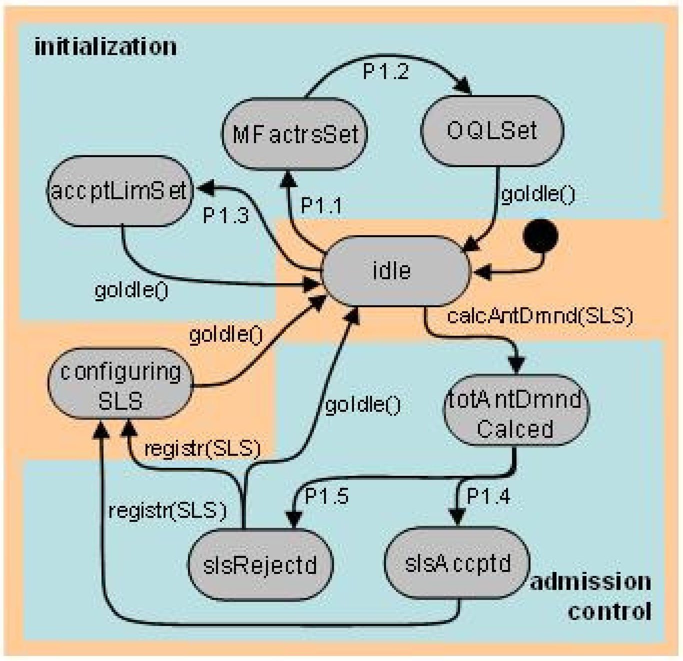


Figure 2-2. SLS module behaviour.

The first two actions in Table 2-1, P1.1 and P1.2, use the notion of service satisfaction and quality levels [2] to set the relevant parameters per QoS Class (QC) and their values range from 0 to 1. Satisfaction parameters essentially define multiplexing factors that are used to derive the rates at which a service is considered *almost* and *fully* satisfied, and quality parameters are analogous to the confidence level with which a SLS is to enjoy the agreed QoS – quality values close to 1 being appropriate for high priority QCs. The example below encodes action P1.1 into policy specification following the Ponder format and defines the almost satisfied factor for AF1 traffic during peak hours.

```
inst oblig /policies/sls-s/P1.1 {
      on    newRPC();
      subj  s = sls-sPMA;
      targ  t = sls-s/servSatisfMO;
      do    t.setAlmstSatisf(af1, 0.2);
      when  duration(08:00-18:00);
}
```

Table 2-1. SLS-S policy actions.

ID	Policy action	Description

P1.1	setAlmstSatisf(QC, Fctr)
	setFullSatisf(QC, Fctr)
		Sets the almost and full satisfaction factors per QC

P1.2	setQltLvl(QC, OQL)	Sets the overall quality level per QC

P1.3	setSUConsrv(Value)
	setSUModrt(Value)
	setSURisky(Value)
		Sets the RAB upper limit in a conservative, moderate, or risky fashion

P1.4	accpt(SLS)	Accepts an SLS subscription request
P1.5	rejct(SLS)	Rejects an SLS subscription request

Subscription admission control is based on resource availability and more specifically on a Resource Availability Buffer (RAB) [2] which holds aggregated traffic demand of subscribed SLSs on a per Traffic Trunk (TT) basis. Policy actions P1.3 set the upper limit – subscription upper (SU) – as a percentage of the RAB for accepting new subscriptions in a conservative, moderate, or risky fashion and define the level of associated risk in satisfying the QoS requirements. The acceptance limit is used as a constraint when deciding whether to accept or reject a request as in the policy example below encoding P1.4, where a request is accepted if the aggregated traffic demand is less than SU.

```
inst oblig /policies/sls-s/P1.4 {
      on    totalAnticipatedDemandCalced(SLS);
      subj  s = sls-sPMA;
      targ  t = sls-s/acMO;
      do    t.accept(SLS);
      when  t.getTotalDemand(SLS.tt) < t.getSU(SLS.tt);
}
```

In contrast to the static nature of the subscription module, the service invocation logic is based on run-time events/ notifications to regulate the traffic entering the network. The policies used here perform dynamic admission control on the number and types of active services, as well as on the volume and type of traffic admitted into the network. The behaviour of the SLS-I module as a result of policies is depicted in Figure 2-3 and the relevant actions supported are listed on Table 2-2.

The run-time operation of the module is triggered by threshold crossing alarms and service invocation requests. The latter activate policy actions for accepting/rejecting a request (P2.4, P2.5), whereas the former initiate a set of actions that control the rates of incoming traffic (P2.8) and change invocation admission control parameters (P2.6, P2.7). These parameters are used as constraints in accept/reject policies and essentially define the treatment of new service invocations: the closer the aggregate value of current TT utilisation and the requesting SLS traffic rate to AC_{max}, the less the chances of the SLS being successfully invoked.

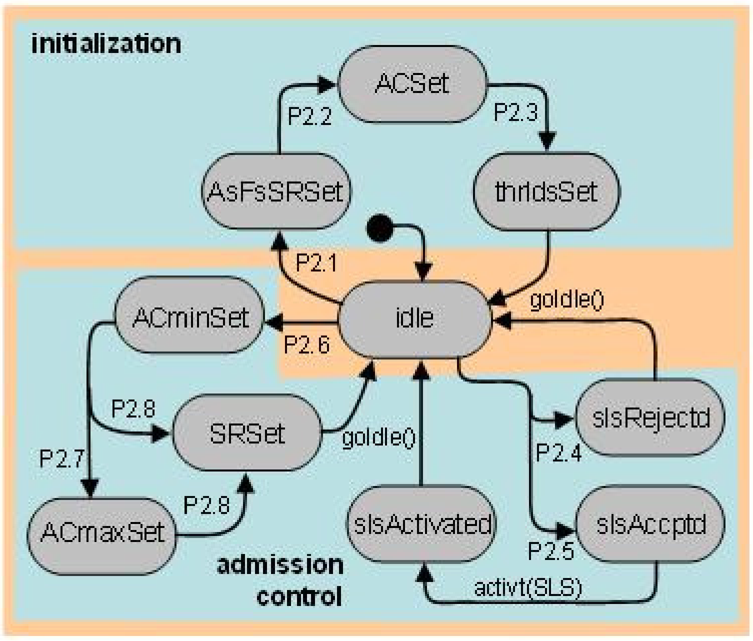


Figure 2-3. SLS-I module behaviour.

Table 2-2. SLS-I policy actions.

ID	Policy action	Description
P2.1	$setSR_{AS}$(TT, Value) $setSR_{FS}$(TT, Value)	Sets almost/fully satisfied service rates per TT
P2.2	$setAC_{min}$(TT, Value) $setAC_{max}$(TT, Value)	Sets admission control min/max limits wrt RAB per TT

P2.3	setTCL(TT, Value)
	setVCL(TT, Value)
		Sets target/very critical level thresholds wrt RAB per TT

P2.4	rejct(SLS)	Reject an SLS invocation request
P2.5	accpt (SLS)	Accept an SLS invocation request
P2.6	$incrAC_{min}$(TT, Value) $decrAC_{min}$(TT, Value)	Increases/decreases admission control min parameter of a TT
P2.7	$incrAC_{max}$(TT, Value) $decrAC_{max}$(TT, Value)	Increases/decreases admission control max parameter of a TT

P2.8	incrSR(TT, Value)
	decrSR(TT, Value)
		Increases/decreases the service rate of a TT

The example policies below encode actions P2.6 and P2.8 to handle the event of a TCL threshold crossing alarm. When enforced, they take proactive measures to avoid potential congestion built-up, by decreasing AC_{min} – thus decreasing the probability of accepting new invocations – and the service rate by 20% and 10% respectively.

```
inst oblig /policies/sls-i/P2.6 {
      on    TCLAlarmRaised(up, TT);
      subj  s = sls-iPMA;
      targ  t = sls-i/servAdjustMO;
      do    t.decrACmin(TT, 20);
      when  duration(08:00-18:00);
}
inst oblig /policies/sls-i/P2.8 {
      on    TCLAlarmRaised(up, TT);
      subj  s = sls-iPMA;
      targ  t = sls-i/servAdjustMO;
      do    t.decrSR(TT, 10);
      when  duration(08:00-18:00);
}
```

2.2.2 Traffic Engineering Policies

Traffic Engineering is responsible for fulfilling the contracted SLSs by deriving the long- and short-term network configuration. The former is realised by Network Dimensioning (ND) that maps the forecasted traffic demand provided by SLS-S onto physical network resources in terms of MPLS Labelled Switched Paths (LSPs) and anticipated loading for each DiffServ QoS class (QC) on all interfaces. The policy actions on Table 2-3 guide the functional behaviour of dimensioning to effectively achieve an optimal configuration in terms of network load.

Table 2-3. Network dimensioning policy actions.

ID	Policy action	Description

P3.1	setNDMin(QC, BW)
	setNDMax(QC, BW)
		Sets min/max allocation per QC

P3.2	setupLSP(QC, [Path], BW)	Sets explicit LSPs per QC through nodes of Path

P3.3	calcHopCountMin(QC)
	calcHopCountMax(QC)
	calcHopCountAvg(QC)
		Derives the hop count constraint of ND algorithm per QC, with different strategies

P3.4	setMaxHops(QC, HopNum)	Sets the maximum number of hops per QC
P3.5	setMaxAltPaths(QC, [TT], PathNum)	Sets the maximum number of alternative paths per TT
P3.6	allocSpareBW( )	Distributes spare BW among QCs
P3.7	redOverBW( )	Reduces over-allocated BW

The first two policy actions, P3.1 and P3.2, allow an administrator to set upper and lower bounds of network capacity per QC, and setup explicit paths through which specific traffic will be routed. Another function of ND is to handle the QoS requirements of the expected traffic in terms of delay and packet loss. This process is simplified by transforming the delay and loss requirements into constraints for the maximum hop count for each traffic trunk. The actions P3.3 allow for different strategies in achieving this objective with *calcHopCountMax* being conservative and appropriate for high priority traffic. The core component of ND is an optimisation algorithm and its objective is to find a set of paths for which the BW requirements of TTs are satisfied and the delay and loss requirements are met by using the hop count constraint as an upper bound. This is a non-linear optimisation problem which is solved by the gradient projection method [3]. Action P3.4 sets an upper bound on the number of hops the calculated paths are permitted to have, and P3.5 defines the number of alternative paths the optimisation algorithm should allow for every traffic trunk that belongs to a specific QC for the purpose of load balancing. In the final stages, ND assigns the residual physical capacity to the various traffic classes or reduces the allocated capacity because the link capacity is not enough to satisfy the predicted traffic requirements. Actions P3.6 and P3.7 can achieve these objectives with different strategies by explicitly, equally or proportionally reducing/distributing capacity between the various traffic classes.

The provisioning directives from ND are treated as nominal values and are a result of predicted demand. Dynamic TE functions are deployed by DRsM that deal with traffic fluctuations around the forecasted values in order to optimise resource utilisation. The DRsM policy actions of Table 2-4 manage available resources based on utilisation monitoring.

Table 2-4. DRsM policy actions

ID	Policy action	Description
P4.1	incrAlloc(Link, QC, BW)	Increases allocation per QC
P4.2	decrAlloc(Link, QC, BW)	Decreases allocation per QC
P4.3	incrThs(Link, QC, BW)	Increases tracking thresholds per QC
P4.4	decrThs(Link, QC, BW)	Decreases tracking thresholds per QC

Utilisation monitoring generates events to trigger policy actions that increase/decrease resource allocation as well as utilisation tracking thresholds using absolute (e.g. in kbps) or relative values (e.g. as a percentage). The example policy below encodes action P4.1 to increase the allocation for AF1 traffic by 10% on a particular link, in the event of an upper tracking threshold crossing alarm. State chart behaviour representations of both TE modules can be found in [4] and [5].

```
inst oblig /policies/drsm/P4.1 {
      on    drsmAlarmRaised(upprTh, link1, af1);
      subj  s = drsmPMA;
      targ  t = drsm/allocMO;
      do    t.incrAllocRel(link1, af1, 10);
      when  duration(08:00-18:00);
}
```

2.3 Quality of Service Management Policy Conflicts

Thus far we have described the behaviour of the individual QoS management modules and provided example policies that would be used to manage these modules. In order to use these policies in a running system it is necessary to check that they do not conflict with policies already deployed in the system. In this section we present the different types of conflict that could arise between policies written for different modules in the QoS management system. We start by providing the taxonomy of the conflict types that have been identified.

2.3.1 Conflict Classification

Although it would be possible to classify these conflicts using different characteristics we have chosen to distinguish the categories based on their level of abstraction, the subsystem in which they occur and their specificity to the application domain as we believe these most naturally reflect the scope in which they occur. First we distinguish between conflicts that are module-independent and those specific to the two management subsystems. Module-independent conflicts may occur among any of the QoS management policies, whilst service management and traffic engineering conflicts are specific to the operations supported by the relevant modules. The latter categories are further subdivided into conflicts relating to policies for individual QoS management modules (intra-module), and to policies applying to different modules (inter-module).

Module-independent conflicts represent the simplest forms of inconsistency that may arise between policy specifications and examples include *redundancy*, *mutual exclusivity* and *QC priority* conflicts. Redundancy conflicts may arise because of duplicate policies or policies with inconsistent action parameters in relation to others and can be detected by syntactic analysis. Mutual exclusion conflicts occur between policies implementing alternative strategies that realise the same goal. Examples of the latter conflict type include SLS-S policies for setting the upper limit in the RAB in a conservative, moderate, or risky fashion, ND policies defining the treatment of spare/over-provisioned BW, and DRsM policies managing the allocation on link resources through different strategies. The various actions are said to be mutually exclusive since there should not be more than one directive specifying an operation on a particular managed resource. An example of such inconsistency would be between a DRsM policy incrementing the resource allocation using an absolute value (e.g. 500 kbps) and policy P4.1 of the previous section. The conflict will materialise if the two policies are triggered by the same event, apply to the same link and QC, and have an overlap in the time constraints.

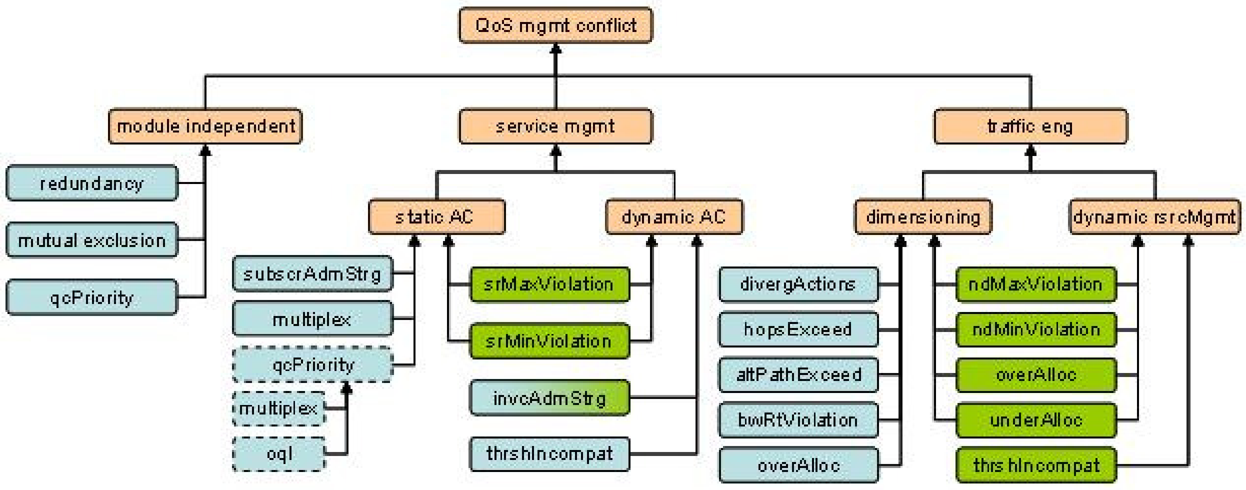


Figure 2-4. Policy conflict classification. Blue and green denote static and dynamic conflicts respectively.

The relative priorities between traffic classes can cause inconsistencies to arise between policies defined on the various QCs in the context of any QoS management module. These are termed *qcPriority* conflicts and will materialise if the effect of a policy action violates the priority between QCs. Figure 2-4 shows two examples of such a conflict between SLS-S policies for setting service satisfaction and quality levels (P1.1 and P1.2). A *multiplex* conflict will occur if the multiplexing factor of a particular QC is higher than that of another QC with lower priority, whereas an *oql* conflict will arise if the quality level of a QC with high priority is lower than that of QC with lower priority.

The above module-independent conflicts, as well as some of the identified inconsistencies, can be determined through static analysis at policy-specification time. Policies governing the behaviour of online modules, on the other hand, are mostly prone to conflicts that can only be detected dynamically at enforcement-time as their manifestation depends on the current state of the underlying managed resources. The next sub-sections describe the various conflicts relating to service management and traffic engineering policies and the conditions under which they arise.

2.3.2 Service Management Policy Conflicts

Conflicts specific to our application domain primarily occur because of inconsistent attribute values set by policies. It is essential that these are individually identified such that the exact reason for their occurrence can be defined and eventually resolved by a network administrator or in an automated manner.

Conflicts related to the SLS-S module can be detected at specification-time and arise between policies governing the process of static admission control. As mentioned previously, the upper limit in the RAB is a major factor for the decision of accepting/rejecting a new subscription request and can be defined with different strategies (P1.3): risky, moderate, and conservative. A *subscrAdmStrg* conflict will arise if the resulting value of the conservative approach is greater than that of moderate, or if the latter is greater than the one generated by the risky approach. *Multiplex* conflicts occur between service satisfaction policies (P1.1) that essentially define multiplexing factors used to derive the rates at which a service is considered *almost* and *fully* satisfied. This inconsistency will occur if the multiplexing factor for fully satisfied is greater than the multiplexing factor for the almost satisfied for the same QoS class, as they are inversely proportional to the service rates produced.

To regulate the traffic entering the network, SLS-I works on guidelines provided by SLS-S. These come in the form of policies that act as constraints and although harmonising the operation of the two modules, they may cause dynamic conflicts that we term inter-module: *srMaxViolation* or *srMinViolation* conflicts occur when service invocation policy actions (P2.8) try to alter the service rate of a specific trunk but violate the almost or fully satisfied rate boundaries provided by the subscription policy P2.1. Besides activating service rate policies, threshold crossing alarms also trigger policies that manipulate AC parameters (P2.6, P2.7) aiming to provide proactive and reactive control over invoked services. The relative position of both thresholds and AC parameters in the RAB of each trunk allows the administrator to adapt the strategy by which services are admitted to the network and potentially avoid the build-up of congestion while maximising resource utilisation. Incorrect definition of these parameters will lead to an *invcAdmStrg* conflict that can occur both at policy specification-time, but also during system execution as AC parameters may be re-calculated on the fly. This inconsistency will arise if AC_{max} is less than VCL, or if AC_{min} is greater than TCL. Lastly, an intra-module static conflict – threshold incompatibility (*thrshIncompat*) – can occur between policies setting the threshold values (P2.3) if TCL – aiming to trigger some proactive measures – is greater than VCL.

2.3.3 Traffic Engineering Policy Conflicts

Conflicts between policies guiding the functional behaviour of Network Dimensioning are static in nature and occur due to contradicting action parameters of BW allocation and routing policies. A *divergActions* conflict may arise between two policies setting the BW allocation boundaries for a specific QC (P3.1), if their parameter values do not converge. These policies may also cause an *overAlloc* conflict if the sum of the allocation corresponding to all the supported QCs exceeds 100%. The same rule applies to explicit actions responsible for the distribution of spare resources or the treatment of over-provisioned BW (P3.6, P3.7). *HopsExceed* conflicts occur if the hop count of the path, through which an LSP is set (P3.2), exceeds the maximum number of allowed hops specified in P3.5, provided both policies apply to the same QC. LSP policies can lead to further inconsistencies – *altPathExceed* and *bwRtViolation* – if their instantiated number is more than the one defined by policy P3.5, for the same QC and TT, or if the specified allocation exceeds the maximum allowed by policy P3.1.

Conflicts related to DRsM policies are mainly dynamic and arise due to ND policies constraining the run-time allocation of resources. These are inter-module conflicts and are a result of the hierarchical relationship between the two modules where policies introduced at the ND level are refined, communicated, and executed by DRsM as well. More specifically, an *ndMaxViolation* conflict occurs when policy P4.1 tries to increment the allocation for a QC but the calculated BW exceeds the upper bound set by the ND directive P3.1. Similarly, an *ndMinViolation* conflict occurs when policy P4.2 tries to decrement the allocation but the calculated BW is less than the lower bound set by the ND. Another high-level directive that is refined down to the DRsM level is a general resource management policy, which explicitly specifies that during a DRsM operational cycle, the full link capacity should be allocated between the various QCs. This implies that a DRsM policy action aiming to increase or decrease the allocation for a specific QC will violate the above rule as the resulting allocation may exceed or be less than the link capacity. We term these inconsistencies as *overAlloc* and *underAlloc* conflicts respectively. The last of DRsM related conflicts is an intra-module conflict and involves policies responsible for the computation of new thresholds and allocation of resources. The inconsistency arises if the allocated BW is below its respective upper utilisation tracking threshold, in which case a threshold incompatibility (*thrshIncompat*) conflict should be signalled.

2.4 Conflict Analysis

The conflict analysis approach presented here has two main aspects: the definition of appropriate rules for determining potential conflicts in policy specifications, and the effective deployment of analysis processes in the context of the managed environment. Detection rules are used to describe the conditions under which a conflict will arise and include information from policies and the managed environment itself to cater for the various QoS management conflicts. Although guidance can be provided by the policy refinement process for some conflict types – in the case of ME conflicts the actions associated with multiple disjunctive sub-goals should be encoded in a detection rule – their specification is a manual process largely relying on the knowledge of an expert administrator. Since most of the defined policies are generic to a certain extent, re-use of the conflict detection rules in other QoS frameworks could be possible. A comprehensive set of detection rules together with system-specific information is used by the analysis processes to determine potential inconsistencies.

The principal challenges in detecting policy conflicts are being able to account for the constraints that limit the applicability of a given policy to specific states of the managed system and the effects of enforcing policies on the states of the managed system. To achieve this, it is necessary to use formal reasoning techniques and formal models of the QoS management system behaviour, policy enforcement mechanisms and the policy rules themselves. Since the QoS management system and the policy enforcement mechanisms concerned are event-based reactive systems, we have used Event Calculus (EC) as the underlying formal representation. In addition to having built-in representations for events and persistence of properties Event Calculus is a suitable formalism because it supports both deductive and abductive reasoning.

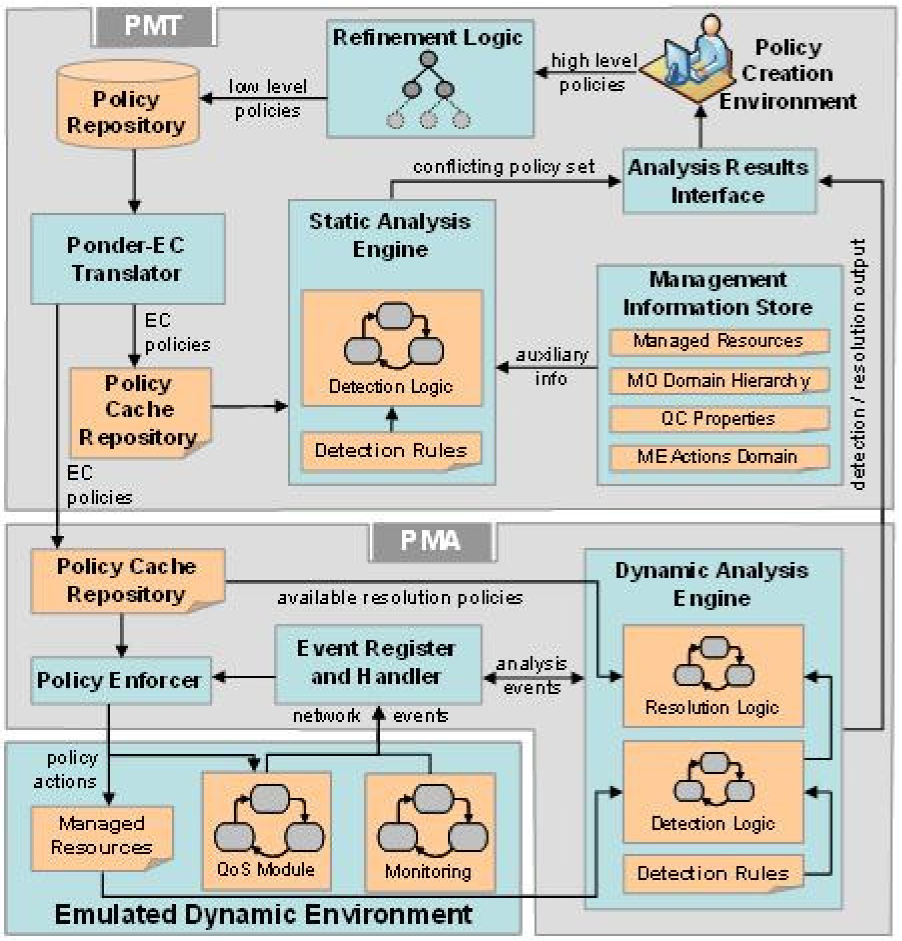


Figure 2-5. Conflict analysis architecture. Static analysis is a centralised process running within the PMT, whereas dynamic analysis is a distributed process integrated within PMAs.

Figure 2-5 outlines the architecture of our approach. Here, we distinguish between static and dynamic analyses as two different processes that are executed at different timescales and physical locations. Static analysis is an integral part of a Policy Management Tool (PMT) and is initiated by a network administrator, whereas dynamic analysis runs in a Policy Management Agent (PMA) and its execution is based on run-time events generated by the network/managed system. Both processes are semi-automated. In the case of static analysis a set of pre-specified rules are used to search the policy space for conflicts whose resolution is manual. Dynamic analysis also makes use of conflict rules, but the invocation of the process and the subsequent detection of conflicts are automated. The run-time resolution process is also automated and is based on a pre-defined set of policies. The details of the two processes are presented in the next sub-sections.

2.4.1 Static Analysis

As shown in the upper part of Figure 2-5, our approach towards static conflict detection is based on the output of a refinement process [6][7], where high-level policy specifications introduced by a network administrator are decomposed into low-level implementable ones and mapped onto their respective EC representation. Before their enforcement, policies are analysed by the static analysis engine, a process initiated by a user administrator, by performing comparisons on a pair-wise basis. This process makes use of specialised detection rules, which are pre-specified by an expert and loaded to the engine, but also domain-specific information regarding the QoS management modules, to identify inconsistencies.

Based on the identified conflict types, we have defined a set of rules expressed in the form of logic predicates that encapsulate the conditions to be met for a conflict to occur. These predicates are used as conflict fluents in EC notation and can be considered as goal states that, when achieved, signify the detection of a conflict. The advantage of using such a methodology is that, in addition to detecting possible conflicts, an explanation as to why a conflict occurred will always be provided.

The conditions for a conflict can either be acquired from the policy specification itself but also from domain-specific information. The example predicate below is based solely on information provided by policies and aims at detecting redundancy conflicts by matching certain key parameters as well as actions in the policy specification through a pattern directed search. The variables of the *conflictData* term are unified during the search if the conditions for a conflict, as defined in the predicate, hold, thus providing all the information pertinent to the conflict.

```
holdsAt(conflict(redundancy, conflictData(PolID1, PolID2, QC1, QC2, OQL1,
OQL2)), T)←
      holdsAt(oblig(PolID1, Subj, op(Targ, Action[Params1])), T) ∧
      holdsAt(oblig(PolID2, Subj, op(Targ, Action[Params2])), T) ∧
      (numParams(PolID1, PolID2) == 1 ∨ numParams(PolID1, PolID2) >= 2 ∧
      intersect(Params1, Params2, Params) ∧ Params \== []) ∧
        PolID1 \== PolID2.
```

Apart from redundancy conflicts the detection of all other inconsistencies requires not only information provided by the policy specification but also QoS-specific information, such as properties of the managed resources and the supported QoS classes. These are encoded in the detection rules as further constraints that should not be violated. The example below represents the relevant predicate for detecting QoS priority conflicts among policies for setting the quality level of traffic classes. This conflict will be detected if the defined quality level of a QC with high priority is less than that of a QC with a lower priority.

```
holdsAt(conflict(qcPriority, conflictData(PolID1, PolID2, QC1, QC2, OQL1,
OQL2)), T) ←
      holdsAt(oblig(PolID1, Subj, op(Targ, setQltLvl(QC1, OQL1))), T) ∧
      holdsAt(oblig(PolID2, Subj, op(Targ, setQltLvl(QC2, OQL2))), T) ∧
      ((QC1::priority > QC2::priority ∧ OQL1 < OQL2)  ∨
       (QC2::priority > QC1::priority ∧ OQL2 < OQL1)) ∧
        PolID1 \== PolID2.
```

The output of the static analysis process is a set of conflicting polices along with an explanation of their occurrence [4]. The resolution of these conflicts is a manual process that has to be carried out by the user administrator, in which policy parameters are modified or, in the case of redundancies, policies are eliminated from the system. A number of conflict resolution methods aiming to automate the process have been proposed in the literature, most of which are based on policy priorities [8][9][10][11]. These allow two potentially inconsistent policies to coexist within the system and involve determining which of the two should prevail in the event of a conflict. This is possible due to the nature of the conflicts considered for which precedence between contradicting policies can be established on the basis of modality, specificity, or recency. The static conflicts identified in this work however, do not allow for automation in their resolution since they mostly occur due to inconsistent policy action parameters rather than just actions. Taking the example of *qcPriority* conflicts, consider two instances of policy P1.2 with which the OQL values associated with EF and AF1 traffic are set to 0.8 and 0.9 respectively. Resolution of this conflict can be achieved by either resetting the OQL value of AF1 traffic to be equal or less than that of EF traffic, or vice versa. Since there is no clear indication as to which of the two strategies to follow, and also because the amount by which the OQL will change, for either, QC has an impact on the overall QoS provisioning objectives, human intervention is unavoidable. The details associated with the inconsistency however, can guide the administrator when correcting it.

2.4.2 Dynamic Analysis and Conflict Resolution

While the analysis process described in the previous section is able to deal with static conflicts, some inconsistencies can only be detected at policy enforcement time as they depend on the current state of the network and the resulting configuration output of on-line QoS management modules (SLS-I and DRsM). For this reason, the process for handling dynamic conflicts needs to be embedded within a PMA which has access to the run-time information required.

Detection of dynamic inconsistencies is still based on a set of pre-specified conflict predicates, which, in this case, require additional information regarding the run-time state of online modules. In the context of our work, the conditions under which a conflict will arise are represented by constraints that depend on the conflict type. The rules for detecting such conflicts are based on the fact that two or more policies violate these constraints. The *conflict(srMaxViolation, …)* fluent below is such an example and indicates the violation of an SLS-S refined directive defining the maximum service rate for a TT. Here, the constraints conveyed to the conditional part of the predicate include the specific policy actions with matching TT parameters, and the actual value of *required* rate calculated by SLS-I. The latter is represented as an argument of the *reqSR* term and the conditions for a *srMaxViolation* conflict will be satisfied if this value exceeds the maximum rate specified by the SLS-S refined policy. Similar rules have been defined for all the identified dynamic conflicts relating to both SLS-I and DRsM policies [5].

```
holdsAt(conflict(srMaxViolation, conflictData(PolID1, PolID2, TT, SR_FS, SR)),
T) ←
     holdsAt(oblig(PolID1, Subj, op(Targ, incrSR(TT, Value))), T) ∧
     holdsAt(oblig(PolID2, Subj, op(Targ, setSR_FS(TT, SR_FS))), T) ∧
      reqSR(TT, SR) ∧ SR > SR_FS.
```

Despite the fact that the resolution of static conflicts is performed manually, this process takes place before policies are deployed in the system and does not impose any run-time overheads on the functionality of on-line modules. Dynamic conflicts however, require system components to both detect and resolve conflicts in real-time, without degrading the performance of the system. This has been the main motivation behind our dynamic analysis approach, which aims at automating the triggering of the detection process and the handling of conflicts at run-time.

Unlike other resolution methodologies [8][9][10], our approach does not involve identifying which of the conflicting policies will prevail based on their relative priority, but provides separate resolution rules that handle potential inconsistencies. These rules are effectively obligation policies which are pre-specified by the administrator using the Ponder format and their triggering events are conflict occurrences rather than network events. Although less automated than precedence-based solutions where resolutions do not require to be defined prior to analysis, our approach is more flexible since: (a) it overcomes the problem where precedence cannot be established, and, (b) custom resolution rules can be provided for handling different conflicting situations that may arise. Once created, resolution policies are translated to their respective EC representations and communicated to the PMA, as shown on Figure 2-5, where they are stored in a local cache repository. They are triggered by events generated by the dynamic analysis engine and their enforcement results in resetting the system into a state in which a conflict is resolved.

It is evident that dynamic conflicts arise as a result of a change in the state of a managed module, which in turn is caused by the enforcement of a new policy. To enable the automatic deployment of the analysis process, the latter needs to be notified of such events. This is achieved by processing the detection rules a priori and extracting information about policy actions that can potentially cause a conflict when enforced. These are encoded in the first field in the conditional part of detection predicates as in the *srMaxViolation* example, and are used to configure the *Event Handler* of Figure 2-5. When the latter intercepts a policy enforcement event matching an action that can potentially lead to a conflict, the analysis process is notified and detection is initiated. If the conflict materialises, the resolution logic of Figure 2-5 enters a *resolving* state which performs a search in the cache repository for a possible resolution pertaining to the detected conflict. If an appropriate resolution policy is identified a notification is passed back to the Event Handler which in turn triggers the enforcement of the resolution.

In contrast to the identified static inconsistencies, one of the two policies involved in a dynamic conflict acts as a constraint, the violation of which is the very reason for the occurrence of that conflict. The use of resolution policies is enabled by the fact that the constraining value can be considered as the "strong" value and is therefore used to quantify the relevant parameter in a resolving policy action. Extending the example of the *srMaxViolation* conflict, the resolution policy below sets the service rate to the maximum permissible value defined by the relevant SLS-S directive. The resolving value, SR_{FS}, can be acquired from the parameters of the SLS-S policy on the fly – as this quantifies the relevant variable in the conflict predicate – and instantiate the associated parameter in the resolution policy action. The latter can be re-used for multiple occurrences of the same inconsistency alleviating the need for human intervention. Furthermore, since the resolution rules are part of the formal description, the analysis engine can determine which resolution policy applies for a particular conflict predicate based on the information provided for that conflict.

```
initiates(sysEvent(conflDetected(srMaxViolation,
      conflictData(PolID1, PolID2, TT, SR_FS))),
      oblig(resPol1, slsiPMA, op(servAdjustMO,
        setSR(TT, SR_FS))), T).
```

As seen, the work presented up to this point provides a comprehensive set of Quality of Service policies for managing Differentiated Services networks, and classify the possible conflicts that can arise between them. Also, we have demonstrated the use of Event Calculus and formal reasoning for the analysis of both static and dynamic conflicts in a semi-automated fashion. The following sections aim at presenting a framework where polices are employed in a different area of Autonomic Management, i.e. firewall configuration management through formal argumentation logic and preference reasoning. Such an approach is complementary to the work described so far, as well as to previous work presented in deliverable D9.3 as part of the first stage of PINIDAM.

2.5 Logic-Based Framework for Representing Firewall Configurations

Despite numerous developments in intrusion detection, application level traffic filtering and other network security techniques, *firewalls* remain the main perimeter protection technique for many corporate or academic organisations. In their most common occurrence firewalls are configured through an ordered set of rules specifying which types of traffic should be permitted or denied based on IP header information. As a wide variety of protocols need to be accommodated and as corporate networks have grown, the number of rules in a typical firewall configuration has been steadily growing.

A firewall implements a significant subset of an organisation's security *requirements* as stated in its *security policy*. Consequently, firewall configurations comprise rules derived from this *high-level security policy*. Tools that can derive firewall configuration rules from higher-level, more abstract specifications reduce the burden on the administrator the size and complexity of the specification and the amount of interpretation necessary to implement the security policy. The process is therefore not only less cumbersome but also less error prone. Because firewall configuration rules implement the organisational security policy, they are often called *firewall policies*, or *policies* for short. We will therefore use the terms *high-level policy* when referring to more abstract, higher level specifications and *firewall policies or rules* when referring to the (low-level) firewall configuration rules.

The firewall administrator's task is to realise the organisational (high-level) security policy using the firewall's (low-level) rule-set. Rubin et al. [12] emphasise both the importance of the correct initial specification of the firewall rules and of their continued maintenance. This is especially true since over time, the firewall rule-set can become untidy and unwieldy, making it complicated and difficult to understand - even to its author. This can lead to the introduction of a variety of errors and conflicts. Al-Shaer et al. [13] present a classification of such conflicts and highlight the likelihood of even an experienced network administrator making errors in the implementation of firewall rule sets. In this section we present a logic-based framework for representing firewall configurations that addresses some of these issues by reducing the burden on the administrator in the following ways: (1) logical abstractions of network elements allow policies to be specified at a higher level of abstraction; (2) automated analysis capabilities allow administrators to determine the effects of changes in high-level policies on the firewall's behaviour; (3) automated rule generation means that administrators no longer have to worry about mapping their high-level policies into low-level rule or the precise order in which the low-level rules must be placed into the firewall. Our approach uses a logical framework of argumentation, within which we map and formalise firewall configurations such that we can perform the firewall policy analysis and rule generation described above. The framework captures the operational semantics of firewall policies declaratively through the logical semantics of argumentation based preference reasoning.

In general, an argumentation framework is a pair $\langle T, A \rangle$ where T is a theory in some background (monotonic) logic, equipped with an entailment relation and A is a binary relation on the subsets of T. These subsets of T form the *arguments* of the framework and A is an *attacking* relation between arguments. For any two arguments A1 and A2 we say that A1 attacks A2 when (A1, A2)$\in$A. The semantics of an argumentation framework is based upon the notion of an admissible argument. These are arguments which counterattack all other arguments that attack them and hence (informally) they are at least as preferred as any other conflicting argument.

Definition 1 (Admissibility of Arguments): Given an argumentation framework $\langle T, A \rangle$ an argument Δ is admissible iff (i) Δ does not attack itself, and (ii) for all arguments Δ', if Δ' attacks Δ then Δ also (counter-) attacks Δ'.

The admissible arguments capture the preferred conclusions of the theory (or policy) and thus give a natural way for capturing preference based reasoning.

We use a particular framework of argumentation as realised by the framework of Logic Programming with Priorities (LPP) and its concrete form of Logic Programming without Negation as Failure (LPwNF) [14][15]. This is particularly suited to representing firewall configurations since the firewall rules and their ordering can be naturally translated into the LPP framework as shown in the next sections. For the purposes of this paper, a *logic program with priorities* in the LPwNF framework denoted by Γ consists of four parts:

(i) a basic part logic program P, consisting of rules of the form:

$$Name : L \leftarrow L_0, \ldots, L_n, \ (n > 0)$$

where $L, L_1, \ldots, L_n$ are positive or negative (classical) literals. (A negative literal is a literal of the form $\neg A$, where A is an atom). As usual in Logic Programming a rule containing variables is a compact representation of all the ground rules obtained from this under the Hebrand universe. Each ground rule has a unique (parametric) name, *Name*, which is a term given at the front of the rule.

(ii) a higher part *H*, specifying conditional, dynamic priorities amongst rules in *P* or *H*. Rules in H have the form:

$$Name : prefer(rule1, rule2) \leftarrow$$
$$L_1, \ldots, L_n, \ (n > 0)$$

to be read: if (some instance of) the conditions $L_1, \ldots, L_n$ hold, then (the corresponding instance of) the rule named by *rule1* has higher priority than (the corresponding instance of) the rule named by *rule2*. The rule itself is named *Name*;

(iii) an auxiliary part *A*, which is a normal logic program defining (auxiliary) predicates occurring in the conditions of rules in *P*,*H* but not in the conclusions of any rule in *P*;

(iv) a notion of incompatibility which, for our purposes, can be assumed to be given as a set of rules defining the predicate *conflict/*2, e.g.,

conflict(rule1, rule2)

which states that (an instance of) the rule named by *rule1* is incompatible with the corresponding instance of the rule named by *rule2*. The incompatibility relation is symmetric and always includes that *L* is incompatible with *¬L* and that *prefer(r, s)* is incompatible with *prefer(s, r)* for any two rule names *r*, *s*.

To represent firewall configurations as an LPwNF theory, we must specify three types of information: the high-level policies, the network elements and traffic types relevant to the policies, and any additional properties to be satisfied when generating the ordering of firewall rules. The symbols used to represent this information in our formalism are shown in Table 2-5.

Table 2-5. Predicate and function symbols for logic-based representation of firewall configurations.

Symbol	Description
ipaddr(Name, [A,B,C,D])	Predicate specifying the IP address A.B.C.D of a host or network Name. Finite domain variables are used to represent address ranges.
traffic(Name, [Protocol, SrcPort,DstPort])	Predicate specifying the protocol, source and destination ports for the traffic type Name. Finite domain variables are used to represent port ranges.
action(Action, Pkt)	Predicate specifying the action (allow or block) to be performed for a packet defined using the pkt/3 function.
ruleorder(Name1,Name2)	Predicate specifying that the rule called Name1 should be given higher priority than the rule called Name2. This is an abducible predicate.
subset(Pkt1, Pkt2)	Auxiliary predicate that is true iff the packets represented by Pkt1 are a subset of those represented by Pkt2. Pkt1 and Pkt2 are defined using the pkt/3 function.
fwrule(Name,Action, Pkt)	Function specifying a firewall rule called Name that defines the Action (allow or deny) for a packet characterised using the pkt/3 function.
pkt(TrafficName, SrcName,DestName)	Function specifying the signature for traffic TrafficName from SrcName to DestName

2.5.1 High-Level Policies

These form the basic part *P* of the LPwNF theory, and specify whether the action of the firewall for particular type of traffic from a given source to destination should be *allow* or *block*. We specify these policies using the *action/*2 predicate which takes as arguments, an action constant (either allow or block) and the packet definition (specified using the *pkt/*3 function). We can use this predicate to define rules for the high-level policy "*allow FTP connections from all hosts in the coyote.com network"* as follows:

Fwrule(coyote ftp reqs, allow, pkt(ftp, coyote, any)) :
  action(allow, pkt(ftp, coyote, any)).

Note that the policy can be specified using names for traffic type, source and destination rather than low-level protocol, port number and IP address information. Also, high-level policies need not specify any rule ordering. The *Name* part of the rule is specified using the *fwrule/*3 function which can be used to identify the firewall rule elsewhere. We use the following rules to propagate the action specified in a given high-level policy to all sub-packets that match the policy:

fwrule(coyote_ftp_reqs, allow, SubPacket) :
  action(allow, SubPacket))←
    subset(SubPacket, pkt(ftp, coyote, any)).

fwrule(tricky_ftp_reqs, block, SubPacket) :
  action(block, SubPacket))←
    subset(SubPacket, pkt(ftp, tricky, any)).

Finally, we use the *conflict/*2 predicate to capture the operational behaviour of firewalls where a single rule is always applied to a packet even if multiple rules match. It is therefore inconsistent for two different rules to apply to the same packet:

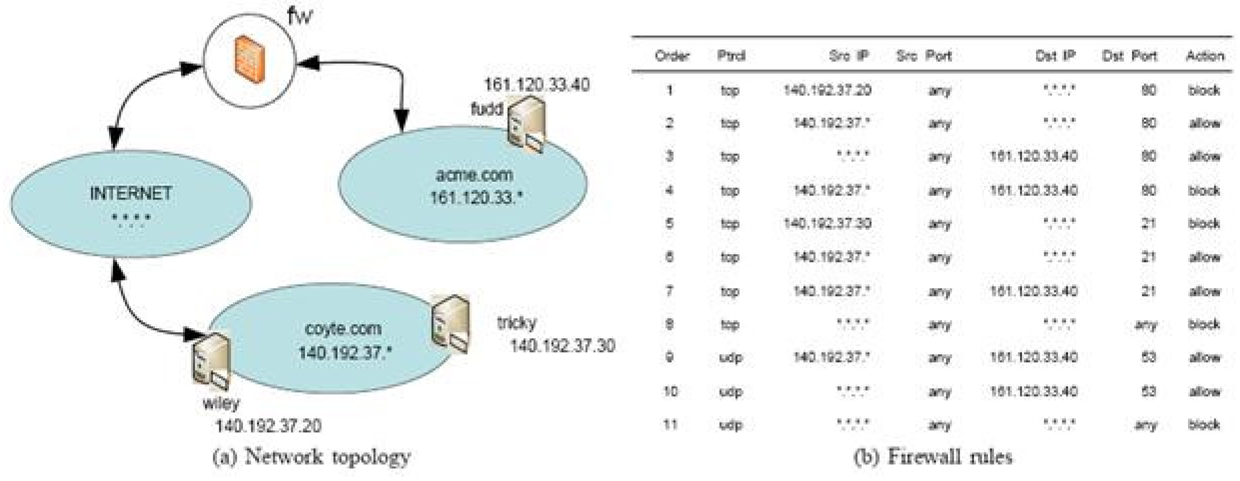
conflict(fwrule(R_x*, , Pkt), fwrule(*R_y*,_, Pkt))* ←
  R_x *6* ≠ R_y.

In Section 2.6 we show how this conflict is resolved in the argumentation reasoning framework by making use of abduction to derive the ordering between conflicting rules. The ordering is computed to match firewall behaviour where the first matching rule is applied.

2.5.2 Network Elements and Traffic Types

This information forms the auxiliary part *A* of the LPwNF theory, mapping the high-level names used in the policies to low-level IP address, protocol and port numbers. We represent both hosts and network IP addresses, ports and protocols for types of traffic and IP packet headers using *ipaddr/*2, *traffic/*2 and *pkt/*3 respectively. Using these symbols, the representation of the *acme.com* network together with the host *tricky.coyote.com* shown in Figure 2-6 is:

ipaddr(acme, [161, 120, 33,D])← D in 0..255.
ipaddr(tricky, [140, 192, 37, 20]).



(a) Network topology

Order	Ptrcl	Src IP	Src Port	Dst IP	Dst Port	Action
1	tcp	140.192.37.20	any	*.*.*.*	80	block
2	tcp	140.192.37.*	any	*.*.*.*	80	allow
3	tcp	*.*.*.*	any	161.120.33.40	80	allow
4	tcp	140.192.37.*	any	161.120.33.40	80	block
5	tcp	140.192.37.30	any	*.*.*.*	21	block
6	tcp	140.192.37.*	any	*.*.*.*	21	allow
7	tcp	140.192.37.*	any	161.120.33.40	21	allow
8	tcp	*.*.*.*	any	*.*.*.*	any	block
9	udp	140.192.37.*	any	161.120.33.40	53	allow
10	udp	*.*.*.*	any	161.120.33.40	53	allow
11	udp	*.*.*.*	any	*.*.*.*	any	block

(b) Firewall rules

Figure 2-6. Example firewall configuration [16].

Additionally, we would represent the ports and protocols relevant to FTP traffic as follows:

traffic(ftp, [tcp, SP, 21])← SP in 1..65536.

We can also specify logical groupings of network elements and traffic types using the *addr_group/*2 and *traffic_group/*2 symbols. For example, assuming we have defined the traffic types *ftp*, *scp* and *tftp*, it is possible to group these together as the type *file_transfer_protocols* as follows:

Traffic_group(file_transfer_protocols, ftp).
Traffic_group(file_transfer_protocols, scp).
Traffic_group(file_transfer_protocols, tftp).

The *subset/*2 relation takes these logical groupings into account when deciding if a particular packet is a subset of another. Finally, we can combine these definitions together with the *pkt/*3 predicate to represent file transfer traffic from *tricky.coyote.com* to *acme* as *pkt(ftp, tricky, acme)*.

2.5.3 Rule Ordering Properties

These properties form the higher part *H* of the LPwNF theory and are used when generating the ordering of the firewall rules. They can be classified into two types, application-independent properties and application-specific properties.

2.5.3.1 Application-independent properties.

An example of an application independent property of the final rule order would be the absence of shadowing anomalies. As shown in [17], a rule R_x is shadowed by a rule R_y if R_x matches a subset of packets matched by R_y, R_x has lower precedence than R_y and the actions of R_x and R_y are different. Therefore to avoid this anomaly, we must make sure that R_x has higher precedence. This can be specified in our formalism as follows:

order(avoid shadow,R_x*,*R_y*) :*
prefer(order(basic,R_x*,*R_y*), order(basic,*R_y*,*R_x*))* $\leftarrow$
  fwrule(R_x*,Action1, Pkt1)* $\wedge$
  fwrule(R_y*,Action2, Pkt2)* $\wedge$
  $R_x \neq R_y \wedge Action1 \neq Action2$
  subset(Pkt1, Pkt2).

2.5.3.2 Application-specific properties

These are properties that are specific to the network configuration and organisation in which the firewall is deployed. For the example in Figure 2-6, the following rule specifies that rules that allow WWW traffic from host *wiley.coyote.com* have higher precedence than rules that block WWW traffic from the *coyote.com* network:

order(www wiley,R_x*,*R_y*) :*
prefer(order(basic,R_x*,*R_y*), order(basic,*R_y*,*R_x*))* $\leftarrow$
  fwrule(R_x*, allow, pkt(www,wiley,))* $\wedge$
  fwrule(R_y*, block, pkt(www, coyote,)).*

In the next section we will show how the logic framework presented above can be used to generate firewall configurations.

2.6 Generating Firewall Configurations

As described previously, generating a firewall configuration entails mapping the high-level security policies to low-level rules as well as deriving the ordering of those rules in the configuration. The combination of the high-level policy and the network information specifications defined in the previous section and deductive reasoning can be used to fulfil the first part of the generation process. However, deriving the rule order requires that we complete the higher part *H* of the LPwNF theory, in order to resolve the conflict that specifies it is incompatible for two firewall rules to apply to the same packet (see Section 2.5.1). To deal with such incompleteness we can use *abductive reasoning* to help us fill in the missing information such that the rule ordering properties we defined are satisfied.

Formally, the integration of abductive reasoning within an argumentation framework is done by first isolating the incompleteness in some of the predicates of the theory, which are called *abducible predicates*. We will denote the set of abducible predicates by A. We then extend the argument rules of the theory, Γ, by a set of generic rules of the form *name : A* for any (ground) literal *A* whose predicate is an abducible predicate. In addition, we add in the theory generic priority rules that give any (existing) rule whose conclusion is incompatible with *A* higher priority over the new argument rule *name : A*. We can then apply the same argumentation base semantics, as described above, allowing us now to include in our admissible arguments missing information in the form of abductive hypotheses supported by these new arguments for abducible literals.

We apply this to the problem of generating a rule order for the firewall policy, by introducing an abducible predicate, *ruleorder(R_x, R_y)*, which captures the abductive hypothesis that rule R_x has higher precedence (i.e. is stronger) than rule R_y:

$$
\begin{aligned}
&order(basic,R_x,R_y) : \\
&prefer(fwr(R_x, Action1, Pkt), fwr(R_y, Action2, Pkt)) \leftarrow \\
&\quad R_x \neq R_y \wedge ruleorder(R_x,R_y).
\end{aligned}
$$

We also include an incompatibility definition to ensure that the orderings *ruleorder(R_x,R_y)* and *ruleorder(R_y,R_y)* cannot be part of the same admissible argument:

$$conflict(order(basic,R_x,R_y), order(basic,R_y,R_x)).$$

This incompatibility definition, together with the rule ordering properties defined previously will constrain the possible abducible hypotheses that can be included in an admissible argument.

We can now generate the rule ordering by invoking the argumentation reasoning procedure with the goal of satisfying each of the firewall requirements specified in the basic part of the LPwNF theory. The admissible arguments generated for these queries will include the *order/*3 terms that specify the relative order between rules such that the defined properties are satisfied. We utilise a simple finite domain constraint expression to translate the relative ordering between rule pairs into a total order for the rule set. In the next section we summarise how our framework would be used in practice.

2.7 Firewall Configuration Framework Usage

The framework we have developed provides a comprehensive environment for firewall configuration authoring and maintenance. In essence, based on the network information, the high-level security policy requirements and the desirable properties for the firewall configuration, the framework can be used in three ways: a) to review a firewall configuration by querying the formal model for reachable nodes, types of traffic allowed/denied etc. b) to analyse a firewall configuration in order to detect anomalies and to detect if deployment specific properties have been violated and c) to generate a firewall configuration that can be translated to the format specific to the firewall used. Whilst a) is typically achieved by using the argumentation framework in a deductive reasoning mode both b) and c) are achieved through using argumentation together with abductive reasoning. The analysis performed in b) takes into account the rule ordering of the actual configuration as shown in detail in [18], whilst in c) a new rule order is being generated.

It is relatively simple to acquire network topology information and to automatically translate it to the formal representation. Security policy requirements can be specified in terms of traffic flow requirements across the firewall. Higher-level concepts such as sub-networks, address groups, service specific port ranges, rule-sets, etc. can be easily defined at the logical level or read from existing configurations. The firewall administrators would thus only need to specify and/or modify these requirements ignoring rule ordering aspects. Desirable properties such as absence of anomalies or deployment specific properties (i.e., specific to the network topology) need to be formally specified. However, these specifications are not likely to change often and would typically be loaded from libraries. These properties are used as goals during the analysis phase where the abductive proof procedure is used to identify counter examples that violate these properties. The same properties are used as background knowledge during the generation phase. Thus, the generated firewall configuration satisfies the required properties. If all properties cannot be satisfied the abductive proof procedure will fail to find any admissible ordering.

Note that the framework remains general. Thus it is possible to specify not only precedence relationships between individual firewall rules but also precedence relationships between precedence relationships. This allows defining how precedence relationships should be defined based on external factors. For example, the framework can be easily extended to have a default behaviour such that precedence is given to rules that correspond to he most frequent traffic types thus optimising the firewall configuration to the expected traffic profile.

2.8 Conclusions

The behaviour of Autonomic Management Systems is bound to be controlled by a number of high-level goals which are certain to result in policy conflicts as they are refined in lower-level policies. Therefore, addressing such policy conflicts is of vital importance if Autonomic Management is to become a mainstream solution to manage complex systems.

The two complementary approaches here presented to address policy conflict analysis are a significant advancement upon current conflict analysis techniques that can be found in the literature. Also, they are a complement and a logical step forward in relation to previous work developed in PINIDAM, thus enhancing the capabilities and improving the suitability of the policy-based framework to support Intra & Inter Domain Autonomic Management.

Our first approach towards policy conflict analysis is based on the formalisation and reasoning provided by Event Calculus and its application in the domain of Differentiated Services Quality of Service management. The subject of the analysis techniques presented here is a set of management policies that can be used to influence/control the behaviour of key modules in the process of QoS provisioning. The various inconsistencies that can arise between these policies have been identified and classified based on their characteristics, which are used to describe the reasons and the conditions under which a conflict will arise. This approach also defines conflicts that can occur between policies applied to a single management module (intra-module), or between policies specified for different modules (inter-module) as a result of their hierarchical relationship, but the main characteristic distinguishing between conflicts is the time-frame at which they can be detected. This has driven the design and specification of two different methods to address the issues associated with the analysis of conflicts that can be detected statically, at policy specification-time, and those that can only be determined dynamically, during system execution, based on feedback regarding the current state of the managed system. These techniques have been implemented and integrated in a conflict analysis tool aiming to provide a network administrator with a usable interface through which to interact with the management system and perform both static and dynamic consistency checks.

Our second approach towards policy conflict analysis is based on argumentation logic and its application in the domain of firewall configuration, thus making it a good complement to our first approach. Argumentation logic permits non-monotonic reasoning with conflicting rules and complex reasoning over the priorities of those rules. This permits to perform both analysis and generation of anomaly free firewall configuration rules thereby significantly reducing the burden and complexity of maintaining firewall configurations. Its advantages over other techniques lie in its modularity allowing customisation to the specifics of the network, the security policy and the properties desired of the resulting configuration, in its flexibility allowing analysis and review through both deductive and abductive reasoning over the specification and in its explanatory power as each logic derivation can be traced and presented to the user. Thus, we believe the above features make our approach an attractive alternative to existing techniques for firewall configuration management. However, these advantages appear to be gained at a significant performance cost (although we have not attempted any optimisations). The performance achieved so far appears to remain within acceptable limits although further evaluation work is needed on larger specifications, which are unfortunately difficult to obtain.

3 Automated Service Management using Emerging Technologies (ASMET2)

3.1 Introduction

Highly available and resilient networks play a decisive role in today's networked world. As network faults are inevitable and networks are getting more and more complicated, finding effective fault recovery solutions in a timely manner is becoming a challenging task for administrators. Therefore, an automated mechanism to support fault resolution is essential towards an efficient fault handling process. The ASMET project is designated to investigate and design a framework to assist administrators to deal with intricate fault recovery tasks based on the emerging technologies such as case-based reasoning, peer-to-peer technology and AI-based automated planning approach.

Our contribution in the previous phase (deliverable D9.3) proposes an architecture to support automated fault recovery in traffic engineering, recovery knowledge discovery and automated recovery planning. Building on the previous phase the research and contributions of the current project phase (ASMET2) are summarised as follows:

- Refinement of knowledge discovery (KD) subsystem. KD is a key subsystem in the proposed architecture, which is designed to discover the relevant recovery knowledge based on the distributed case-based reasoning approach. The objective of KD is to provide automated planning (AP) subsystem with necessary recovery information by the planning process.
- A survey on the applications of the AI-based automated planning in the network and IT management is carried out in order to gain a better overview on the state-of-the-art, which is essential for our endeavour towards an enhanced design and a better integration of the automated planning approach to the fault recovery management.

3.2 DARING – Distributed Case-Based Reasoning System

DARING has a P2P architecture, where peers share resources with other peers and provide search facilities for other peers. A peer usually acts both as a server to answer queries received from other peers and as a client to route queries to other peers. Moreover, a peer also operates like an independent CBR system [19]. It reasons on a query and a local case database to propose answers. The architecture of a typical peer thus becomes plentiful with several modules related to case processing, case maintenance and a CBR engine; whereas peer learning, ranking and selection are embedded in query processing. DARING also exploits semantic search mechanisms associated with CBR engines to improve case retrieval and case reasoning.

Since a DARING peer involves several functions including operating a CBR engine, processing cases and queries, and communicating with peers, it needs sufficient bandwidth and processing power. We consider a backbone network of super peers as an appropriate overlay for DARING. This overlay sticks to the characteristics of Gnutella-style super-peer networks [20]. Such networks organise peers into several clusters, any cluster contains peers connected to a super peer, and the connections among super peers form an unstructured overlay network. The overlay includes a feedback scheme that improves the performance of the flooding search mechanism. We focus on the feedback scheme for DARING in the rest of our discussion.

3.2.1 Feedback Search Scheme

CBR systems usually learn a case from verifying the proposed solution on a real testing platform. Implementing such a platform in DARING is problematic, thus CBR engines cannot verify their proposed solutions. This information, however, is crucial for CBR engines to update their knowledge for better reasoning. Moreover, peers use this information to assess the expertise of peers (i.e., which peers are good in resolving a specific problem) and similar queries (i.e., which peers recently solved problems similar to a specific problem) for improving the search mechanism.

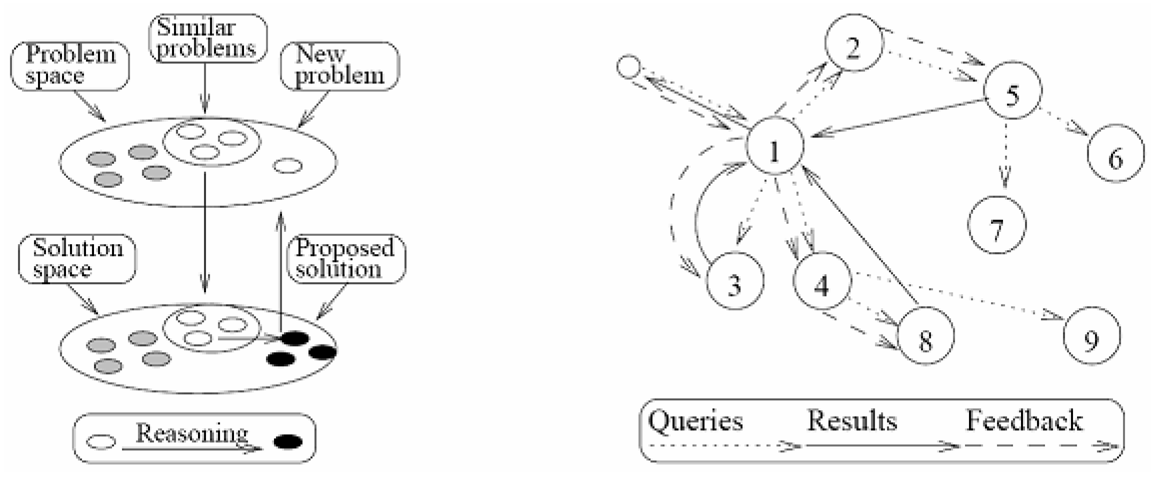


Figure 3-1. Case-based reasoning (left) and the proposed P2P feedback scheme (right).

We introduce a feedback scheme that takes on the task of evaluating the answers of peers and providing the evaluation to related peers. In particular, a peer receives several answers from other peers for its query; the peer uses its CBR engine to rank the answers either with or without the testing platform and instructions from operators. The evaluation results are sent back via the peers routing the query to the answering peer (following the query route). An example is shown in Figure 3-1 where peer (1) sends a query, peers (3), (5), (8) return answers, and peers (2), (5), (4), (8), (3) learn from the feedback. This feedback scheme allows peers to learn the correct answers and who the expert peers are, thus selecting the proper peers for subsequent similar queries.

3.2.1.1 Peer Learning

In the following, we assume that peers maintain an expertise value for other peers. The following algorithms update the expertise of peers and the information of similar queries, which can be used for the peer ranking and selection for the forthcoming queries. The information of a query comprises peers that have good answers for the query. Upon receiving feedback, a peer updates the list of peers *p* and their expertise values *e* using the following formula:

$$e_n = \begin{cases} \beta & \text{if } p_i \notin UpdLst \,\&\, p_i \in FbkLst \\ e_o + \dfrac{\alpha r}{k(k+1)} & \text{if } p_i \in UpdLst \,\&\, p_i \in FbkLst \\ e_o - \dfrac{\alpha}{k(k+1)} & \text{if } p_i \in UpdLst \,\&\, p_i \notin FbkLst \end{cases}$$

FbkLst is the list of *k* feedback peers in the feedback message and *UpdLst* is the list of peers related to recently similar queries; *r*=1,…, *k* is the grade of answers; *β* is the initial expertise value; *α* is a heuristic factor. According to Algorithm 1, each peer's expertise value in *FbkLst* will be increased by *αr/k(k+1)* with 1≤*r*≤*k* if the peer is in *UpdLst*, or initialised to *β* otherwise. Other peers in *UpdLst* will be decreased by *α/k(k+1)*. This update avoids not only reducing the expertise values of peers unrelated to the query but also changing the expertise values rapidly, which may lead to dominating peers. For each update, *k* feedback peers totally increase *α/2* except for initial peers; *α* is thus chosen between 0.4 and 0.6 for this purpose. As this algorithm depends on loops at lines (2) and (4), (6), and (12), its worst case complexity is *O(nnl+k+nl)* where *n*=|*QryLst*|, *k*=|*FbkLst*|, and *l*=***max***{|*PrLst$_i$*|}. Note that in the average case, we assume *k* is much lower than *n* and that only very few queries will be similar, leading to *O(n+l)*.

Algorithm 1: Peer expertise update

```
   Input: $QryLst$: a list of elements $\{q,PrLst\}$;
   $\{q,PrLst\}$: a query & a related list of peers;
   $\{q_f,FbkLst\}$: a query & a list of $k$ peers;
   $ExpLst$: a list of elements $\{p,e\}$;
   $\alpha,\beta,\epsilon$: parameters;
   Output: $ExpLst$
1  $UpdLst \leftarrow \emptyset$
2  for each $q_i \in QryLst$ do
3    if sim$(q_f,q_i) > \epsilon$ then
4      $UpdLst \leftarrow UpdLst \cup PrLst_i$
5    end if
6  for each $p_i \in FbkLst$ do
7    if $p_i \in UpdLst$ then
8      $\{p_i,e_o\} \leftarrow \{p_i,e_o + \frac{\alpha r}{k(k+1)}\}$ in $ExpLst$
9    else
10     add $\{p_i,\beta\}$ to $ExpLst$
11   end if
12 for each $p_j \in UpdLst$ do
13   if $(p_j \notin FbkLst)$ then
14     $\{p_i,e_o\} \leftarrow \{p_j,e_o - \frac{\alpha}{k(k+1)}\}$ in $ExpLst$
15   end if
```

Algorithm 2: Query information update

```
   Input: $QryLst$: a list of elements $\{q,PrLst\}$;
   $\{q,PrLst\}$: a query & a related list of peers;
   $\{q_f,FbkLst\}$: a query & a list of $k$ peers;
   $\epsilon$: parameter;
   Output: $QryLst$
1  found $\leftarrow$ false
2  for each $q_i \in QryLst$ do
3    if sim$(q_f,q_i) > \epsilon$ then
4      $PrLst_i \leftarrow PrLst_i \cup FbkLst$
5      move $\{q_i,PrLst_i\}$ to the top of $QryLst$
6      found $\leftarrow$ true
7    end if
8  if not found
9    add $\{q_f,FbkLst\}$ to the top of $QryLst$
10 end if
```

Figure 3-2. Peer expertise update algorithm (left) and query information update algorithm (right).

Upon receiving feedback, a peer also updates the list of recent queries by comparing the query q_f to each query in *QryLst* using the similarity function ***sim***(q_f, q_i). If the value returned from ***sim***(q_f, q_i) reaches the threshold ε, the element *i* is updated with the related peers. Otherwise, a new element is added to *QryLst*. The top elements of *QryLst* indicate the most recently used queries. In addition, there may be more than one element updated for one feedback depending on the similarity threshold. As this algorithm mainly depends on loops at lines (2) and (4), its worst case complexity is *O(nlk)*. Note that in the average case, we assume *k* is much lower than *n* and that only very few queries will be similar, leading to *O(n+l)*.

Peers use a least recently used (LRU) policy to maintain the lists of expert peers and recent queries. This policy reduces the size of these lists. As a result, several queries and peers can be removed if a peer is either short on storage or processes many queries in a short time frame.

3.2.1.2 Peer Ranking and Selection

Peer ranking considers two aspects to rank peers: the similarity of queries and the expertise of peers. The peer learning process regularly updates the two local lists *QryLst* and *ExpLst*. The ranking algorithm, in general, looks for the best peers (i.e. expert peers) which answered similar queries. Note that the peer expertise update algorithm decays the expertise of inactive expert peers. The algorithm uses the similarity function to compare a given query with each query in *QryLst* in order to obtain a ranking list *RnkLst* of peers and their expertise values. The *RnkLst* is sorted by the expertise of peers. The algorithm may return an empty set if the algorithm fails to find similar queries.

The selection algorithm chooses *k* promising peers from the ranking list, the query list, and the neighbour list. It first selects up to *k-1* peers from the ranking list. If not enough ranked peers are available, it considers the top elements of the query list, which includes the most recently used peers. It then fills the selected list of peers with random neighbours. The neighbour list is updated constantly following the change of the network; whereas, the other lists help to select peers that likely provide good answers. The neighbour list is needed when the other lists cannot recommend any peers. More precisely, the algorithm always adds at least 1 random peer from the neighbour list that is different from the already selected γ-1 peers in order to add a random factor to the search mechanism.

3.2.1.3 Evaluation of Feedback Search

We focus on the simulation and performance analysis of the heuristic search mechanism using the feedback scheme described above (FB search for short). Experiments compare the performance among the FB search, the flooding search mechanism (FD search} for short) and the random search mechanism (RD search for short). The FD search simply chooses n_c neighbours (also number of connections per peer) from the neighbour list; whereas the RD search randomly chooses peers to forward queries using the same number of peers as the FB search does. Since the feedback scheme is application-specific, we cannot compare the FB search to other approaches.

We have considered two metrics to evaluate the performance of these mechanisms: the average number of messages per query and the recall rate of retrieved documents. The first metric emphasises the reduction of the number of messages exchanged, which is very crucial in DARING because query processing in CBR engines may require significant computational resources.

However, sending fewer messages may lead to missing many relevant resources. The second metric, therefore, measures the capability of retrieving relevant resources. We have employed the FD search as the exhausted search that provides an upper bound of the recall rate, and the RD search that provides a lower bound of the recall rate. The purpose of using the RD search is to verify the performance of the feedback scheme in the FB search; i.e., using the same number of peers, the FB search tends to choose the expert peers. We also examine if the FB search can reach the recall rate limit obtained by the FD search. We have simulated a Gnutella-style network whose topology is established by random connections. We have used the CACM (3204 titles) and SIMILE (354 titles) bibliographic datasets (available at http://www.cs.utk.edu/~lsi/ and http://simile.mit.edu/wiki/Category:Dataset) to test the search mechanisms. Each title in the dataset is described in the field-value vector that is suitable for using the above similarity function. The parameters were configured as shown in Table 3-1.

In the first experiment, we count the number of messages in the FD search and FB search. For the FD search, the number of query messages is calculated based on the number of selected neighbours and different churn rates *r*=0.05, 0.1 and 0.15. With n_c =4 and *ttl*=4, the maximum number of messages per query is $(4^4+4^3+4^2+4^1)$=340. For the FD search, a query consists of query and feedback messages. The number of feedback messages depends on the number of peers that have correct answers. As this number is small and CBR engines do not process these messages, the maximum number of messages per query is $(3^4+3^3+3^2+3^1)$=120 with n_c =3 and *ttl*=4, thus reducing 65% messages.

Table 3-1. Simulation Parameters.

Notation	Value	Description
N_p	100	Number of peers
N_q	200	Number of queries
n_c	4	Number of connections per peer
n_d	35	Number of documents per peer
R	0.05, 0.1, 0.15	Churn rate
Ttl	4	Time to live
Α	0.4	Parameter
Β	0.5	Parameter
Γ	3	Parameter
Ε	0.8	Parameter

In the second experiment, we create three query sets for both datasets: (i) 200 queries with 50% similarity; i.e., 100 original queries and 100 duplicated queries; (ii) 600 queries with 70% similarity; (iii) and 900 queries with 85% similarity. Note that the FB search only shows its efficiency with similar queries and we only consider two duplicated queries as similar queries. The experiment was run on 100 super peers with 3500 titles. We compute the recall rates of the FD search, the FB search and the RD search. We assume that relevant documents mean correct answers, so we can use the number of relevant documents to assess the expertise of peers, and obtain the accurate performance of the FB search.

Figure 3-3 plots the comparison of the recall rate among three search mechanisms using the query set of 50% similarity. The rate of the FD search remains 0.79 or 0.80 from the 80th query. The FD search seems to reach the recall rate limit that depends on the precision of the similarity function. The rate of the FB search increases as the number of queries increases. The FB search performs better than the RD search that only stays at a rate of 0.4, thus proving that the feedback scheme improves the efficiency of the FB search. The FB search and RD search behave similar in the first 70 queries because peers learn queries and update the list of expert peers in this period, thus the ranking and selection algorithms cannot provide expert peers and work as the random algorithm.

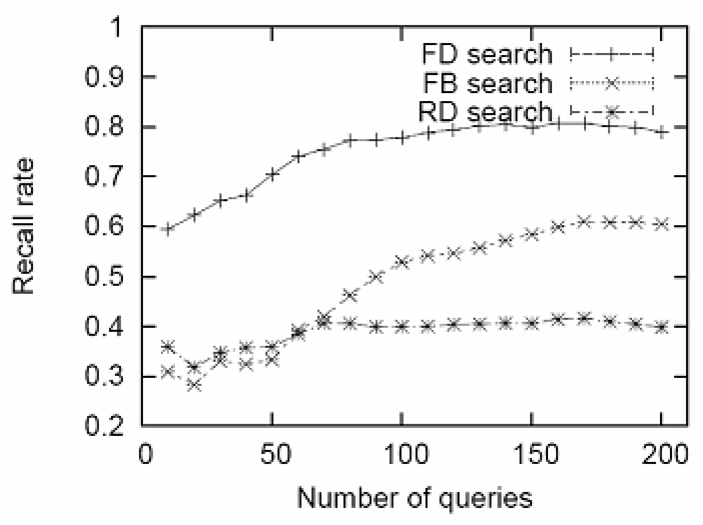


Figure 3-3. Recall rate evaluation for the data set with 50% similarity.

We intensify the experiment to the second and third query sets of 70% and 85% similarity. The result shows the increasing recall rate of the FB search with various degrees of similarity; the recall rate grows quickly from 20% (0.39) to 50% similarity (0.6), and slowly from 50% (0.6) to 85% similarity (0.72) because peers cannot learn better peers with many similar queries. With the query set of 85% similarity, the FB search reaches 90% (~0.72/0.80) of the recall rate limit and reduces 65% of the number of messages per query as described above.

3.2.2 Related Work

Research on heuristic search mechanisms in unstructured P2P networks has led to several different approaches. Every peer in *routing indices* [21] contains a table of aggregate information of neighbours and resources which allows the peer to send a query to promising routes. *Profiles* for peers in [22] record recent queries and answer peers by monitoring the *query* and *queryhit* messages. The approach to cluster peers based on contents has also been exploited in many studies. The *firework query model* [23] defines topics and uses special links to divide peers into groups. The *guided search* [24] proposes *guide rules* to form sets of peers whose contents are semantically similar. The *topic-segmented* overlay [25] employs a *topic-driven* query routing protocol to forward queries to proper areas of the network, where the matching documents reside. The dynamic topology approach [26] proposes a*cquaintance* links that connect a peer to a set of peers sharing the common interests. Although exploiting different information about queries, resources and peers, all these approaches have in common that they employ an efficient topology to improve the routing scheme in heuristic search mechanisms. This work shares the same idea with these approaches; however, we examine a problem-solving application scenario where peers cannot be clustered easily by their contents and successful queries cannot be learned directly from answers.

3.2.3 Short Conclusion

This work of DARING includes the feedback scheme and the search mechanism to search for relevant resources. The algorithms of peer learning, peer ranking and peer selection explain how a peer updates the expertise values of peers and the information of similar queries. Briefly, the querying peer, upon receiving the answers, evaluates these answers and sends feedback about good answers to the answering peers for forthcoming queries. The experiments show that the feedback search significantly reduces the number of messages exchanged, but still obtains a high recall rate. In addition, the feedback search performs better with query sets of high similarity. This work has been published in [27].

3.3 A Survey on Applying AI-based Automated Planning to Autonomic Management

In this section we present our survey on applications of the automated planning technique in the autonomic network and IT management. For the sake of completeness, we not only survey the applications in fault management, but also investigate those areas such as change management, configuration management, Grid computing and pervasive computing.

3.3.1 Automated Planning

3.3.1.1 Theoretical Background

Planning is a process of selecting and organising of actions to achieve the designated objectives. Automated planning is a study of such process computationally [28]. It is one of the classical research fields of artificial intelligence which is concerned with using well-established searching and reasoning techniques to generate plans according to the current state of the underlying dynamic system and the pre-defined state.

Algorithmically, a planning mechanism has as input a set of actions, a predictive model of underlying dynamic system and the planning goal. The output to a planning problem is one or more courses of actions that fulfil the goal. A planner operates on the set of actions and elaborates a plan which transits the current state of the system to the goal state. Advanced planners also include the capability to schedule the plan executions.

Conceptually, a planning problem could be denoted as P$=(\Sigma, s_0, g)$, where Σ is a representation of state-transition system and s_0 denotes the current state of underlying dynamic system, g represents the goal state. A state-transition system is an abstract model of planning system and is defined as a triple $\Sigma=(S, A, \gamma)$, where S is a set of states $\{s_1,\ldots,s_n\}$ and A=$\{a_1,\ldots,a_n\}$ is a set of actions to be chosen from. γ is a state-transition function, which takes the current state s_n as well as a selected action a_n as input and transits the system to the next state. In order to assist reasoning process by action selections, each action $a_n, i \in 1,\ldots,n$ in the action set is supplied with further descriptions: $a_i^{pre} \subseteq S$ and $a_i^{post} \subseteq S$. The first notation a_i^{pre} denotes the preconditions of this action, which are conditions that have to be fulfilled before this action can be chosen. a_i^{post} describes the post conditions or the effects of a particular action. In some cases, a cost function $c(a_i, s_i)$>0 is also needed to evaluate costs of the action a_i associated in the state s_i. This function is meaningful when more than one option is available for planner to choose from. For example, the costs involved for recovering a web server with reboot or with switching a standby system are surely different. The cost could be, for example, financial cost or time costs etc., according to the application scenarios.

A solution to a planning problem P is a sequence of actions $a_1,\ldots,a_k, k \leq n$, which takes system from initial state s_0 to one of the pre-defined goal states. The state-transition function γ maps actions to the new state, $s_1 = \gamma\ (s_0, a_1) \ldots\ s_n = \gamma(s_{n\ 1}, a_n)$. Provided with the above described information, a planning algorithm finds a plan by searching in the space of possible states configurations. Different planning algorithms utilise various methods to generate a plan.

3.3.1.2 Types of Planner

Since the incarnation of automated planning research, different types of automated planners have been developed. Thanks the increasing interests of researchers on automated planning methods, the efficiency of planners have been significantly improved during last few years. Planning problems which planners are capable to solve are shifting from experimental toy problems to the complex real-world planning activities including mission-critical applications such as tasks planning in Mars rover and military combat operations. Despite the rich diversity, the existing planners could be categorised into following types:

- *State-Space Planner*. It is the simplest planner which based on state-space search algorithms. The search space is a subset of the observed system states. The planner elaborates to find applicable actions that transit the current system state to goal state. *Forward searching* and *backward searching* algorithms could be applied. Although simple, state-space planner suffers from the efficiency problem as the search space grows. Without assistance of suitable heuristics, it searches through the complete state space in order to construct a plan.
- *Plan-Space Planner.* Unlike the state-space planner, the plan-space planner does not operate on the state-space but in the plan-space, a plan here means partial plans, it contains a set of actions and retains the casual relationship and orderings between them. One of the obvious advantages that the plan-space planner possesses is that, this approach significantly reduces the search space, further more, the built plan is partially ordered. The associations of time and resource constraints are much more straightforward.
- *Planning Graph Planner.* This type of planners includes a very unique and powerful search space called *planning graph*. To find a plan, the planner initially creates a graph with ***n*** levels until solution is included in the level, in this case a goal state, is reached. Each level ***i*** contains all actions whose preconditions and applicable constraints are satisfied in the level ***i-1***. After the graph is constructed, backtracking approach is mostly used to find a path between goal states to the initial state. The path found is the solution to the problem. Experiments show that planning graph planner is much faster than previously mentioned two types of planners.
- *Hierarchical Task Network (HTN).* The HTN planner takes the different approach as other planners. Instead of searching for a solution in the way as above-mentioned planners do, the planning procedures are done by iteratively decomposing high-level tasks into subtasks until primitive tasks have resulted. This type of planner takes *tasks* and *methods* instead of actions as inputs where tasks include *primitive* and *non-primitive tasks* and methods describe decompositions of non-primitive tasks. Methods could be regarded as *recipes* which integrate sophisticated knowledge by solution-finding. HTN is the most widely used planner in solving real-world planning problems, because it is orders of magnitude more efficient than other planners.

3.3.2 Applications of Planning Techniques in IT Management

Having introduced the basic of the automated planning, in this section we give an in-depth discussion on the autonomic applications integrated with planning techniques. With the growing maturity of the planning research, more and more researchers [29], [30] in autonomic management realm have identified this emerging technology as opportunity to increase the degree of autonomy of their management applications and systems. The conducted survey investigates automated planning related research efforts ranging from planning of IT management activities, including configuration management, fault management and change management, to planning of workflows of grid computing and pervasive computing.

3.3.2.1 Configuration Management

IT services and underlying systems need to be properly configured to meet the objectives of businesses and individuals. Modern IT services usually consist of huge amount of subsystems and components, both software and hardware. Manual configurations of such complex, multi-components systems are not only costly but also error-prone tasks. The automated planning in the context of configuration management could give operators flexibility to automatically find proper configuration actions and synthesise configuration workflows. Planning system should have the ability to transform the high level configuration goal into specific, executable workflows. During the planning process, it should also consider applicable configuration policies, in order to guarantee that the solution is conformed to the management policies.

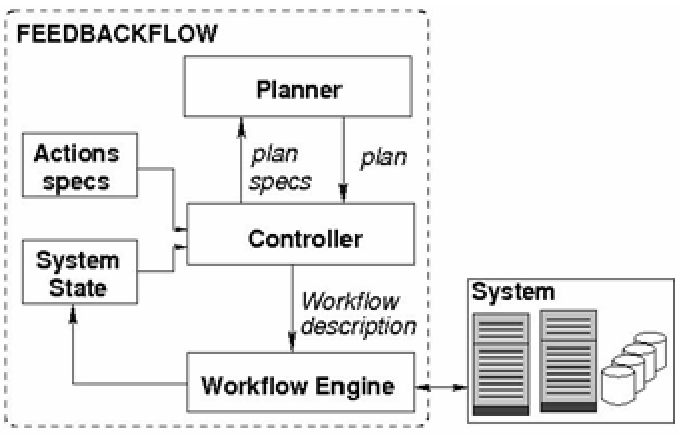


Figure 3-4. Architecture of FEEDBACKFLOW.

Andrzejak et al. [31] design a framework called FEEDBACKFLOW, which incorporates the automated planning component to compose complex management tasks. As its name suggests, this framework is based on the closed loop of control with feedbacks, including planning, executing, validating and re-planning. As Figure 3-4 shows, the FEEDBACKFLOW framework consists of a planner, a controller and a workflow engine. The controller is responsible for taking the declarative action specifications from operators; those include atomic actions such as the installation of certain type of software components or set parameters to certain value. Further more, the controller also take system description from workflow engine. The description of the system is based on its states, which is denoted with PDDL syntax [32]. The controller triggers the consecutive steps based on aforementioned inputs. The planner finds applicable configuration plan based on the goal specification and the system description. The format of generated plan is translated by the controller into XML and forwarded to the workflow engine for execution. Each workflow has a state manager unit, which keeps tracking the execution status of actions, and a stop unit designated in case a non-recoverable failure takes place. A unit represents the selected atomic actions is also included in a workflow. Once the execution terminates, the system manager unit updates the controller, it decides either repeats the planning cycle or terminates the whole process.

The planner in this framework is based on the Model Based planner (MBP) [33]. Different from other planners introduced in the previous section, MBP synthesises plans in terms of symbolic model checking, in which propositional formulae is used to represent finite state automaton. The advantage of using MBP is that, it not only does plan composition, but also includes plan validation and plan simulation, where generated plans are verified for their correctness according to the given property and their executions could also be simulated on a given domain. Moreover, MBP also works with a partially observable planning domain, which means the status of domain is only available after executions of action during run-time. In terms of configuration management, we could not expect that status of planning domain is fully observable during the planning phase; some sensing actions need to be performed. Although MBP is claimed to be a highly efficient planner, integration in FEEDBACKFLOW shows that it still suffers from the critical performance limit. The prototype of FEEDBACKFLOW only works well with small examples. The experiment shows, it takes prohibitive long time to compute a complex plan, which makes the practical value of the framework questionable. The major bottleneck of FEEDBACKFLOW, we believe, lies in the planner, where both domain model and plans are constructed with symbolic model checking. Despite the performance improvement against explicit model checking, it is still time consuming approach
The work from Arshad et al. [34] called *Planit* targets specifically on the optimal software deployment and reconfiguration issues. The motivation behind *Planit* is to cope with intricate software configuration tasks, where many inter-related factors such as time constraints, state of application need to be considered during the process. The conventional configuration methods, which assume a static deployment environment and static plan, are not effective and scalable. The proposed approaches deals with such dynamic based on the process called *sense-plan-ac*t. The proposed architecture adopted a domain of reconfigurable systems, whose model consists of three kinds of entities: component, connector and machines. Components are software entities, connectors describe communication links, and machines are where components and connectors are located.

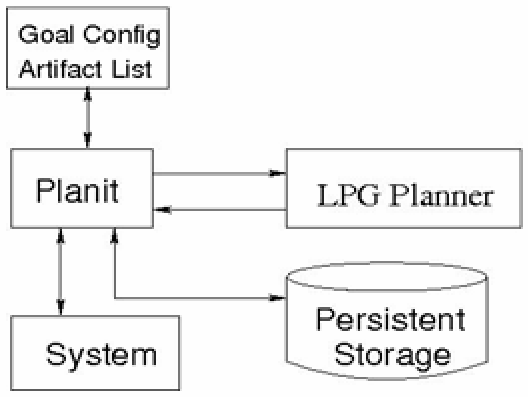


Figure 3-5. Architecture of Planit.

As Figure 3-5 shows, the architecture of *Planit* is much like that of FEEDBACKFLOW. As the mediator of the architecture, the *Planit* component consists of two core sub-components, the *Reconfiguration Manager* and *Problem Manager*. Upon receipt of system events from sensor agents, the reconfiguration manager delegates them to the problem manager, where the current system state and the goal state are derived. The goal state is computed by checking the explicit and implicit configuration, which denote a non-specific predicate of the system after plan is executed and information on detailed described artefacts on configuration. Together with the domain description file, the *Planit* component sends derived initial and goal states to the external planner. Note that all input files to the planner is written in PDDL conform format. The *Planit* relies on the LPG planner [35] for plan syntheses, it searches iteratively for better plans, and plans are judged by pre-defined metrics, such as time. The reconfiguration manager selects the best plan for execution on the system. The persistent storage is used to store contingency plans, which cannot be changed during the life of the system.

The LPG planner is based on local search on the planning graphs to solving planning problem. It enjoys reputation of fast planning speed and has the ability to find multiple plans, with which same configurations objective could be achieved with different costs. This feature provides administrators with flexibility to consider alternative plans if the best plan fails during the execution.

Planit still suffer from several drawbacks, for example, plan execution is done in batch fashion. In the real-world scenario, underlying systems states are changing constantly, the plan made several minutes ago could be invalid already, therefore, a plan and execution interleaving model would be more realistic. Despite fast speed shown by experimental results, *Planit* still suffers from scalability problem. The performance bottleneck, as it by FEEDBACKFLOW, is on the planner. The external LPG planner is taken without any application-specific optimisations, for example, searching space could be reduced using constraints. Those constraints could be derived from management policies.

3.3.2.2 Change Management

IT systems need to be changed in order to adapt to accommodate the needs from users. Change activities include applying fixes or updating of software and hardware in order to enhancing the performance of a target system as whole. Change management is a process to manage those activities. Planning is one of the critical steps involve in change management in a highly complex and inter-dependent IT environment. Challenges regarding automating change planning involves the consideration of dependencies of systems and components, incorporating policies in the planning process and capture/reuse the best practices or change receipts. An effective change plan should optimise the change duration and cost. In the meanwhile, minimise the impact of changes related incidents. We look into several applications which address those challenges with automated planning feature.

Fink [36] proposes a generic architecture for integrating various information sources necessary for IT change planning. The Hierarchical Task Network (HTN) planning paradigm augmented with temporal constraints is used to model and to hierarchically refine the high-level change tasks into consistent change plans, with consideration on involved infrastructure and policies or constraints.

The IT knowledge base is one of the key components included in the proposed architecture. It acts as mediator between planner and various knowledge resources commonly available in enterprise, such as Change Management Database (CMDB) [37]. All of the aggregated information is stored as objects in the database, include a snapshot of current operational state of systems and components that need to be changed. Besides aggregating divers knowledge resources, the knowledge base prepare query operations for change planner to access and to manipulate data.

Another cornerstone of the architecture is the change planner, which utilises aggregated knowledge to synthesise change plans. In addition to the knowledge provisioned by knowledge database, it also needs to query the policy repository to find applicable change policies to guarantee the conformability of the composed plans. As mentioned before, the change planner is built based on the HTN planning paradigm, whose details is introduced in the section II.

The major characteristic of HTN planning paradigm is its hierarchical decomposition of high-level tasks into atomic tasks (i.e. non-decomposable, executable tasks). To serve this purpose, HTN planning incorporates so-called *methods* to iteratively decompose tasks into subtasks until atomic actions are achieved. Those methods could be used to encode receipts or best practises of management knowledge. The obvious advantage of HTN planner compare to classical ones, such as state-space or plan-space planners, is that HTN planner allows sophisticated knowledge representations and reasoning capabilities through using of decomposition methods. The change planner in the proposed approach integrates *change catalogue repository* which stores change template and best practices, or in HTN terms, tasks, methods and operators in order to refine the high level change requests into executable change plans.

A change plan usually consists of not only sequential but also parallel activities and each activity is commonly associated with time duration. To make an efficient change plan, those issues need to be considered during the planning process. To this end, the proposed approach incorporates temporal reasoner to serve this purpose. It maintains *a Simple Temporal Problem (STP)* [38] data structure, in which temporal information is encoded as time constraints and the minimal equivalent constraint network is computed. During the plan refinement process, temporal constraints are continuously added to the STP, every plan has start and end time point represents the duration of the plan, as tasks are decomposed by methods into subtasks, new temporal constraints are added to enforce the ordering structure of the sub workflows as encoded in the decomposition methods. Although not specifically discussed, the approach from Fink leaves an interface in the design to policy repository, which is to be queried during the planning process, to check the policy conformability of the intermediate plans.

To summarise, it is an interesting approach to apply HTN based planner into change management. The rich knowledge representation through the use of decomposition methods and tasks makes this planning paradigm suitable for the real-world applications. In fact, many HTN based planner are applied to solve real-world problems, such as activity planning in the mars rover and fire emergency evacuation applications. The temporal reasoner component assists planning process by guaranteeing efficient and consistent plan structure. Unfortunately, the approach is only tested against a set of relatively small change planning instance, the performance of the proposed planning approach on real situations is not provided. Moreover, although provided with interface to policy repository, how planner and other components integrating policy knowledge into planning process are only addressed marginally.

3.3.2.3 Fault Management

Fault management encompasses activities such as detection, diagnosis and recovery of faults. The fault recovery as the last step of fault management requires that recovery solutions are quickly found and recovery steps are efficiently organised. Current recovery approaches are mostly done manually and they are inefficient and error-prone. To this end, automated planning technique can be applied to reason on the recovery steps. During the planning process, it is also possible to associate the constraints and policies in order to guarantee the conformability of the discovered solution plans.

Arshad et al. [39] made the initial attempt to integrate automated planning technique into fault recovery in distributed system. The discussion in the presented work focuses on the domain description and representation of the system states (including current and goal states). The domain knowledge is described using a pseudo-PDDL language. It includes three parts: *predicates, functions and actions*. Predicates specify the states of each component in the system, e.g. *ServletEngineStarted(servletEngine)* or *ApplicationServerWorking(applicationServer).* Actions are operations that could be executed in the system. Each action is specified by set of pre-conditions and post-conditions, which are set of logical expressions. They describe under which conditions an action could be selected and how the action is expected to influence the current state of the system. An action could only be selected if the preconditions are fulfilled. Further more, each action can be specified with duration, depicting the necessary execution time duration needed for this action. The duration is defined as a function in the functions list. With this construct time oriented metrics could be associated during the planning process. A further work [40] from the same authors investigated the replanning issue, i.e. what could be done if the plan composed by planning mechanism fails during the execution.

Much as other planning based approach, Arshad et al. use an available planner [35] for the plan synthesis, no customising or adaptation work is done toward the improvement of performance. As the planner is the same as the one which is applied by *Planit,* we will not give further details here to avoid the reiteration. As the experimental tests are done on a small scenario with a very limited set of components, the question regarding the scalability of the proposed approach is left open.

3.3.2.4 Grid and Pervasive Computing

The work from Ranganathan et al. [41] discusses the application of AI planning technique in the pervasive computing environment. In such an environment, multiple components with different functionalities and computing powers are involved. To achieve certain objectives, a user needs to find out the combinations of those components which fulfill the requirements. Considering the number of participating components in the pervasive environment, this process could be challenging for users. In the same time, the dynamic and fault-prone characteristics of a pervasive environment decrease its usability. Based on this consideration, Ranganathan proposes a planning framework called *Gaia*, which seeks to improve the usability and user friendliness of the pervasive environment by applying the features of automated planning.

Basically, the proposed framework takes the abstract goal specification from users and decides how those goals are to be achieved. The framework applies Blackbox planner [42], which unifies the graph-planning approach to with a SAT-based approach. The description of planning domain and planning problem is based on the STRIPS (Stanford

Research Institute Problem Solver) [43] description, which is a subset of PDDL. A user is expected to give a set of goal specifications and the corresponding goal state is generated by the planning algorithm. During this process, the planning algorithm applies the utility functions so that the goal state with maximum utility could be chosen. As soon as the goal state is determine, the STRIPS planner tries to find a solution from current state to the goal state. If no solution to the chosen goal state is found, the next goal state with lower utility will be tried. Otherwise, the solution is executed by calling the methods of services, devices or applications with proper parameters.

Grid computing provides its users with a powerful platform for large scale computation tasks and data storage to solve scientific problems. The targeted users are scientists working on high energy physics, biology or astrophysics etc which require the extreme high computational power and the storage capacity. Grid is built on the geographically distributed computing resources, which is transparent to its users; however, performing computational job in a Grid environment is still highly complex tasks, which require the users possess expertises in high performance computing, networking and operating systems. Such complexity prevents the target users from using Grid, because of lack of required skills. Current Grid middleware does not provide much support to users by selecting and configuring suitable resources and components. To address such problem, Blythe et. al. propose a knowledge based approach to generate scientific workflows with helps of automated planning [44][45]. The work flow generation tools augmented by the automated planning technology enables scientists composing and allocating computational tasks automatically. The key characteristics of the suggested approach are the use of application metadata and component input & output, explicit representations of constraints and policies. A user could provide specific goals in terms of metadata with constraints and the planning mechanism finds a solution accordingly. To build a flexible knowledge representation scheme, an ontology based approach is used to allow the system to evaluate the suitability of given resources to provide the particular services. The suggested approach is embedded in the Laser Interferometer Gravitation Wave Observatory helping researchers to compose scientific workflows, experiment shows that a feasible plan contains over hundreds of components could be found in few seconds.

3.3.3 Discussion and Research Challenges

In this part of the reported work, we survey a wealth amount of management applications, which are augmented by automated planning techniques. The surveyed applications show that the automated planning is a flexible approach to be integrated into autonomic management systems. Its abilities on task decomposition, reasoning and constraints processing give it the potential as an enabling technology to increase the degree of autonomy in the future management systems.

We also found that, despite the recent proliferation, many surveyed integration efforts are still in their infancy. To pave the way for the better integrations, we identify several key research issues and summarised as follows:

- *Admissibility of plan knowledge description language*. Almost all of the surveyed applications use PDDL or the subset of PDDL languages to represent domain and planning knowledge. Despite its expressiveness, PDDL is a language specific to planning research community and is not admissible to the IT operators and administrators. To make the automated planning technique more accessible to network and service management realm, a more admissible knowledge representation scheme is need.

- *Customising and optimising of planners*. All of the surveyed applications take available planners in their original forms. No efforts on adaptation and customisation are made; we believe this is responsible to the performance issues which haunt most of surveyed applications. To this end, the application specific customisation and optimisation need to be made in the future researches.
- *Integration of constraints and policies*. The surveyed applications only accept very rudimentary constraints, for instance, time costs of the particular actions. In a real world management scenario, much more constraints and policies need to be considered in order to make a feasible plan.
- *Perceiving state of underlying dynamic system*. A planning system need to correctly perceive the current state in order to make the right choices on actions. The frequent fluctuation in an underlying system may render the plan made in few minutes ago unusable; therefore, a planning system must be capable to constantly perceive the current state of the system in order to adjust its planning process.

4 Context Architectures for Autonomic Management (CARAM2)

Next generation networks are becoming service-aware. This implies that context coming from various sources can be used for supporting service management operations and also for network resource optimisation. As a consequence, the complexity for managing context information grows with abundance of services and diverse service requirements placing new requirements on information control and management as well as on services and network resources usage. The CArAM research project aspires to find a solution for providing a functional concept for context information gathering in different provider domains involving questions of modelling, exchange protocols, accounting, inconsistency and resilience on behalf of future autonomic communications. The use of ontologies for support and management of services (OSM) will facilitate the transition from service agnostic management architectures to service and network resource-awareness by proposing the functional components following autonomic principles.

The common denominator for CArAM work is the use of ontologies as a mechanism for semantic integration events signalled between the involved domains. The first subsection presents a brief overview of the work done in the area of creation and modelling of services in order to provide service assurance and management functions. It is followed by the description of the context processing middleware using ontologies.

4.1 Management and Context Integration based on Ontologies for Pervasive Services Operation in Autonomic Communication Systems

This work proposes the semantic enrichment of management information described in information and data models with ontological data to provide an extensible, reusable common manageability semantic plane. The proposed semantic plane provides new tools to integrate context information with management service operations, and offers a more complete understanding and hence, a more inclusive governance of the management of resources, devices, networks, systems and services for promoting the integrated management with a common information model. The semantic plane is made up of the Ontology for Support and Management (OSM), as well as the ontology-based functional framework (Onto-CONTEXT framework). OSM is designed to provide the necessary semantic richness to represent the different types of information to need to be integrated in network management operations, using a formal methodology that uses available context data as values used in various service management operations. OSM augments the expressiveness of a policy information model approach by adding domain-specific context data for users, locations, devices, applications, and services. OSM integrates network management information, policy information, context information, and ontology concepts and their relationships to manage services and management operations of those services for NGNs and pervasive systems.

In addition this work defines a generic framework and architecture, as well as specific realisations of each, for network management. The generic framework offers concepts, relationships, and rules for the development of ontology-based management platforms and systems. We introduce context processing based on ontologies in Section 4.2. In Section 4.3 we present service lifecycle and service management operations. Section

4.4 describes our ontology for support and management (OSM) and Section 4.5 concludes our work.

4.2 Context Processing based on Ontologies

For the adaptation of services to the current situation of a user, the services are in need of specific context information. The acquisition of context in highly dynamic environments is a complex process as the appropriate context sources are not known in advance. Moreover, to realise Mark Weiser's vision of ubiquitous computing, many services on the one hand and a good deal of context information on the other hand have to be combined. Hence, we follow a middleware approach to automate context retrieval for services. For the exchange over domain boundaries, services in need of and services offering context information have to agree on a common description of the information. Therefore, a flexible and extensible information model is a basic requirement. This section describes in detail the integration of those two important foundations of context-aware computing.

4.2.1 Introduction

In highly dynamic environments with a multitude of mobile entities, it is important for (i) context-aware services to find and for (ii) context information services to provide context information to many other systems. A restaurant finder service for example looks for venues close to the user's location and may depend on other information describing the user's situation. Which context information services, i.e. context sources it has to query will not be known until the actual query is made. On the other hand, context information about any entity, as for example the user's location can be used in many different context-aware services. To relieve context-aware services of the intricacies of context retrieval and composition as well as to facilitate the reuse of context information at the same time, in [46] we proposed infra-structural services, namely the CoCo Infrastructure. Context information is exchanged over domain boundaries, thus all involved actors have to agree on how to express and interpret context information. Therefore, an information model has to balance expressiveness and inferential efficiency. In [47] we introduced a modelling approach based on ontologies that takes into account the special characteristics of context information. In the next sections we will go into details about the integration of this ontology-based information model into our context middleware.

4.2.2 The CoCo Infrastructure

The CoCo Infrastructure (see Figure 4-1) acts as a broker between context-aware services (CAS) that request context information and context information services (CIS) that provide context information. It therefore relieves the context-aware services from the burden of discovering context information services, data transformation, or derivation of high-level context information from low-level context information.

CoCo stands for *Composing Context*. The CoCo Infrastructure does not only support the request for a single piece of context information (like 'the current position of Alex') but also the request for composed context information (like 'the temperature at the place where Alex currently is'). This requires the ability to describe such compositions. Therefore, we have developed a language that describes the request for composed context information in so-called CoCoGraphs [46]. The graph-like structures are expressed in XML.

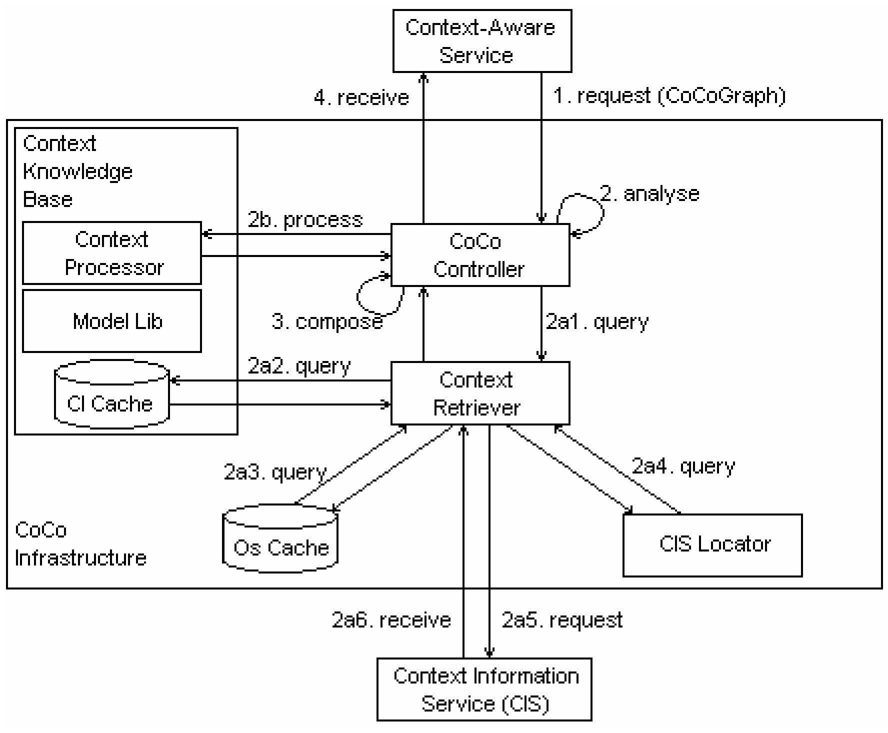


Figure 4-1. Middleware: the Coco infrastructure.

Basically a CoCoGraph is made out of two types of nodes: *Factory Nodes* describing the requested piece of context information and *Operator Nodes* containing instructions how to process one or several pieces of context information. The procedure is as follows: first a request in form of a CoCoGraph is sent from the context-aware service to the CoCo infrastructure (step 1) where it is analysed by the CoCo Controller (step2).

For every Factory Node it sends a request for context information to the Context Retriever (step 2a1), which at first queries the Context Cache to check whether it already has got the requested information (step 2a2). In case it is not available there, the CIS Cache is asked whether it already knows an appropriate context information service to retrieve the information from (step 2a3). If not it then instructs the CIS Locator to find appropriate context information services (step 2a4), to which the retriever sends related context requests. After having received context information for each request (step 2a5 and step 2a6) the retriever matches this information against the request of the controller, selects the most appropriate piece of context information and returns it to the controller.

For each Operator Node, the controller instructs the Context Processor to execute its operation (step 2b), e.g. adaptation, selection or aggregation. Context Processor and Context Cache are part of the Context Knowledge Base, which also includes the Model Lib. The Model Lib contains the ontology-based information model the middleware is based upon; this will be discussed in the next section.

4.2.3 The Context Meta-Model

To facilitate a common understanding and a uniform representation of context, we developed an information model as foundation for context interoperability. This model was introduced in [47]. Context information poses special requirements on an information model as literally all information can be used as context information. It is therefore not sufficient to have a single context model but we propose a Context Meta-

Model (CMM) that can be used by application developers to design their own application specific context models, reuse existing models and combine both possibilities. We base our information model on ontologies and thus gain the possibility to perform reasoning and an important formal basis. Description logics are a set of logic-based formalisms used to specify ontologies. They identify a subset of first order logic that offers a good trade-off between expressiveness on the one hand and determinable and efficient inference on the other. To account for flexibility, the CMM incorporates rules that are based on Horn formulae.

Basic building blocks for representing (context) knowledge in our context meta-model are entity classes, datatype classes, and properties with their associated quality classes (see Figure 4-2):

- Entity class: base construct for representing a group of entities (persons, places, things, events etc.) that belong together because they share some properties.
- Datatype class: base construct for representing a datatype (temperature, noise level, position etc).
- Property: base construct for representing a type of relationship between an instance of an entity class and an instance of either an entity or a dataype class. An example for a property as a relation between two entities on the model layer is: Person ”owns” MobilePhone. Person ”hasPhoneNumber” PhoneNumber relates an entity with a datatype (more details are given in the next subsection).
- Quality class: base construct for representing specific quality aspects of dynamically acquired information (certainty, precision, resolution etc.) also known as Quality of Context.

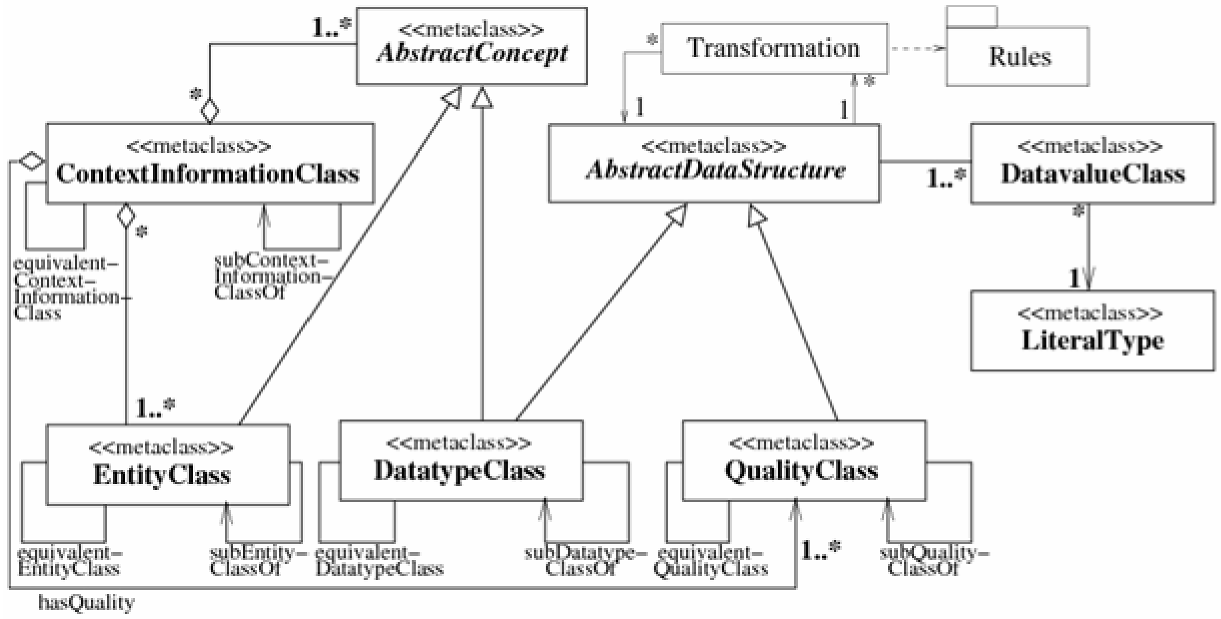

1..*
<<metaclass>> AbstractConcept
*
Transformation
Rules
*
1
1
*
<<metaclass>> ContextInformationClass
<<metaclass>> AbstractDataStructure
1..*
<<metaclass>> DatavalueClass
*
1
<<metaclass>> LiteralType
equivalent-Context-Information-Class
subContext-Information-ClassOf
*
1..*
<<metaclass>> EntityClass
<<metaclass>> DatatypeClass
<<metaclass>> QualityClass
equivalent-EntityClass
subEntity-ClassOf
equivalent-DatatypeClass
subDatatype-ClassOf
equivalent-QualityClass
1..*
subQuality-ClassOf
hasQuality

Figure 4-2. The Context Meta-Model.

In order to represent temporal history information, for every property the acquisition time is captured as a time stamp. It is a mandatory quality class for every property. Dependencies between properties are expressed as rules in the form of Horn clauses. Each rule expresses an implication between an antecedent and consequent: whenever the conditions specified in the antecedent hold, the conditions specified in the successor must also hold. This allows for the specification of consistency conditions as well as derivation rules. Conditions can reference entity classes and datatype classes as well as properties and their associated quality classes. This way a rule can take into account quality information and also specify the quality of the deduced properties.

In addition, there are two special constructs for the semantically rich specification of datatypes: datavalue properties and transformation rules. A *Datavalue Class* is a base construct for specifying data structures, i.e. datatype classes and quality classes. Each datavalue class associates a data structure (*Abstract-DataStructure*) with a literal type and thus allows for the composition of complex data structures from literals. E.g. the coordinates for a position are composed from longitude and latitude. The *Transformation* is the base construct for representing a transformation from values of one data structure to values of another data structure. An example is the transformation between position in Gauss-Krueger coordinates into a WGS-84 format, the transformation function itself is given or described in form of a rule on class level in the *Rules* and the identifier for the rule is given in the model itself.

Further modelling constructs are specialisation-relations that may be specified between two classes of the same kind in order to organise them in (separate) specialisation hierarchies. Finally, there are equivalence-relations. Their semantics is that the first node represents the equivalent concept or role as the second node and should therefore be interpreted equivalently. They are useful for mapping context models that have been developed separately in order to enable interoperability.

4.2.4 Integrating the Context Meta-Model into the CoCo Infrastructure

As context information services and context-aware services as well as a context composition middleware will be operated by different providers, who may not even know each other directly, the main goal behind the implementation was to enable the use of different context models nearly automatically to facilitate interoperability. This can only be achieved if the middleware does not care about the specific context model it deals with at the moment. CoCo itself operates in most parts just within the structure given by the context meta model (CMM). This allows an easy integration of new context models into CoCo on-the-fly at runtime without even stopping the service. Furthermore it should enable us to delegate the retrieval of context model specific code to either a third-party service or maybe even use in Java integrated mechanism.

The identification of a specific context model and *EntityClass* or *Context-InformationClass* of this model is performed on the XML layer by the names-pace and the name of the XML tag, i.e. by its qualified name. The Model Lib of CoCo therefore has an integrated mechanism to translate this qualified name into a Java package name and class name to actually load the correct Plain Old Java Object (POJO).The only prerequisite for this to work is, that the needed classes have to be within the classpath of the CoCo service. This is right now done by copying the JAR packages there but should be replaced by the possibility to automatically load the classes via the Internet directly from the vendor’s location.

In the following subsections, we will first give a detailed view on the process that is executed in the CoCo middleware; we then describe the integration of the ontology-based information model CMM into the CoCo Infrastructure and show the achievements of this work.

4.2.4.1 Parsing and Binding

The first action that takes place when the service is invoked is that it tries to parse the submitted CoCoGraph and determines if it is syntactically and semantically correct, as far as CoCo is able to understand the semantics. Parsing is done in a two step approach, i.e. in the first step we create a DOM tree out of the XML document and do the syntax checks here. The second one is to parse this DOM tree and translate the elements to the appropriate POJOs, e.g. a factory-node DOM element creates a Factory Node Java object. Afterwards one of the most important actions takes place, i.e. the different Input-, Factory-, Operator- and Output- Nodes are bound to each other according to the dependencies specified by the user. E.g. the Factory Node which is responsible to retrieve the temperature for the location where Axel is, is bound to the Factory Node which gets Alex's location as this location is a prerequisite for the other one.

As Factory Nodes normally are just bound to one other node Operator Nodes may be bound to a theoretically nearly infinite number of other nodes. Due to the structure of Operator Nodes there are also bindings within the Operator Node itself, e.g. the output of an Operator Node may be the outcome of a calculation done inside the node or may be a fixed value depending on the outcome of the calculation. The power and possibilities of Operator Nodes are not completely visible at the moment as they depend closely on the context models available.

4.2.4.2 Starting and Running the Nodes

After the binding step is complete the CoCo Controller searches for these nodes which are not bound to any other and starts them. Afterwards, the Node tries to fulfil its task, i.e. either invokes the CoCo Retriever to fetch context information in case of a Factory Node or hands over to the CoCo Processor in case of an Operator Node. In both cases the involved components report either the successful execution or any error to the Node which has invoked them. In case of success the result is returned, i.e. the outcome of the operation or the retrieved context, and the Node is then responsible to inform all nodes which are bound to it about the fulfilment of the task. If an error occurred during execution, e.g. if there is no way to retrieve the requested context information for whatever reasons, the Node is responsible to inform the CoCo Controller about this problem which afterwards has to deal with this issue.

This mechanism goes on as long as there are any nodes left that need to be executed. In case the node is an Output Node it either has the value already in case of a fixed value, or it retrieves it from the node it is bound to and informs the CoCo Controller that it is ready to return its value. After all output nodes have reported to the CoCo Controller it is its task to compose the XML document which is returned to the user.

4.2.4.3 Integrating Jena with the CoCo Infrastructure

The Jena framework [48] is a Java framework for developing Semantic Web [49] applications. It implements the modelling languages RDF, RDFS and OWL, and provides a rule-based inference engine. The Jena database system [50] uses the JDBC to connect to a relational database like MySQL, Oracle or PostgreSQL. RDF triples are stored with subject, predicate and object and each line corresponds to one RDF statement. Jena allows to manage different RDF models simultaneously by assigning an own triple table to each of the models. Jena also includes a SPARQL engine (SPARQL Protocol and RDF Query Language) [51]. SPARQL is a data-oriented query language that searches the model and returns relevant information as a graph or a set of variables. Its syntax is similar to an SQL statement and supports four different request types. Jena is therefore well suited for introducing our semantic information model to the CoCo Infrastructure and allows us to store context information persistently in a database while retaining the semantics as it supports OWL DL.

To integrate Jena with the CoCo Infrastructure, we added an interface ContextInformationCacheJena to the CoCo Infrastructure that includes the procedures insert and query. The Jena database subsystem uses the JDBC driver to connect to a PostgreSQL database. After connecting to the database, a persistent model has to be created with the ModelFactory. Whenever the Context Retriever receives a request for context information, it first queries the Context Cache via the new interface. The query procedure proceeds in five steps: First, from a list of all available models the appropriate has to be chosen and opened. Next, an ontology model is created from the model: the ontology model also contains specification information regarding the ontology language, reasoner and storage location. A SPARQL query searches for the entity and its relevant context. As an entity can have multiple identities, it has to be looked for each of them. When the right entity has been found according to its identity, the sought-after context information can be retrieved in a next step. The result is a model that has to be converted to a DOM element. Usually the DOM parser should be able to convert the model to a DOM element. Unfortunately, the parser could not resolve namespaces, so in our case the query procedure calls a conversion procedure and the model is first converted to a JDOM element and then to a DOM element. The DOM element is finally returned to the Context Retriever. If the query fails because the context information was not in the Context Cache, the Context Retriever looks for an adequate context information service and queries it. The response is then stored in the Context Cache via the insert procedure. The insert procedure has an EntityClass object as parameter. This object is first converted to a DOM object and then stored as an RDF model in the database.

4.2.5 Related Work

Various approaches to infrastructural support of CASs as well as to context modelling exist. In previous works on those topics, existing approaches to context provisioning [46] and context information models [52], [53] have been evaluated thoroughly. In terms of expressiveness, there is no approach that captures all features of context information so far. Most of the existing approaches restrict their generality by stipulating semantic categories and almost none provide constructs to express meta information which is crucial to determine whether the given context information is useful for a particular service. Many of the approaches, in particular earlier ones, lack a formal foundation that is necessary to enable efficient inference, extensibility and distribution of models. In addition, support for interoperability is not explicitly given. Shortcomings in terms of expressiveness and structure result in difficult applicability of an approach in practice.

4.3 Service Lifecycle and Service Management Operations

In the context of managing communication systems that continually are increasing in complexity, the integration of context information for network and services management constitutes a real challenge. It is highly desirable that the context information and operands are accessible from their corresponding context information sources, and that they can be distributed across the network to satisfy service and user requirements in a cross-layered environment in a consistent, coherent and formal way.

Currently, the TMF is specifying many of the management operations in networks for supporting services [54] [55], in a manner similar to how the W3C specifies web services. However, a growing trend is to manage the convergence between networks and services (i.e., the ability to manage the different service requirements of data, voice, and multimedia serviced by the same network), as well as the resulting converged services themselves. The management of NGN pervasive services involves self-management capabilities for improving performance and achieving the interoperability necessary to support current and next generation services.

Self-management features depends on both the requirements as well as the capabilities of the middleware frameworks or platforms for managing information describing the services as well as information supporting the delivery and maintenance of the services. The representation of information impacts the design of novel syntax and semantic tools for achieving the interoperability necessary when NGN resources and services are being managed. Middleware capabilities influence the performance of the information systems, their impact on the design of new services, and the adaptation of existing applications to represent and disseminate the information.

The vision of self-management creates an environment that hides the underlying complexity of the management operations, and instead provides a façade that is appealing to both administrators and end-users alike. It is based on consensual agreements between different systems (e.g., management systems and information support systems), and it requires a certain degree of cooperation between the systems to enable interoperable data exchange. One of the most important benefits of this agreement is the resulting improvement of the management tasks and operations using such information to control pervasive services. However, the descriptions and rules that coordinate the *management* operations of a system are not the same as those that govern the data *used* in each management system. For example, information present in end-user applications is almost exclusively used to control the operation of a service, and usually has nothing to do with managing the service.

4.3.1 The Service Lifecycle from the Management Perspective

In this work, the use of policies for service management is augmented with the use of formal ontologies [56]. This enables the management systems to support the same management data to accommodate the needs of different management applications through the use of rich semantics, some of which are being specified in the ACF. Service management applications for pervasive systems highlight the importance of formal information models in policy-driven service architectures. The policies are used in the managing of various aspects of the lifecycle of services. It is important to identify what is meant by the term “service lifecycle” in the framework of our work. Thus, the following sections describe the pervasive service lifecycle as a set of stages, including its complete customisation, management, deployment, execution and maintenance.

This section describes how to use formal models for facilitating management tasks based on the use of context-awareness, which is used both in service provisioning as well as in the modelling of the service lifecycle modelling for pervasive services. Hence, context information provides the management systems with an extensible foundation of the necessary information to execute the service management operations in complex environments. This is realised using an organisational view that enables the semantics of the service lifecycle to be explicitly modelled and managed. This in turn ensures information interoperability necessary to manage different services in pervasive applications. This section describes the organisational view for the pervasive services lifecycle, which can be divided into six distinct phases with specific tasks as depicted by Figure 4-3.

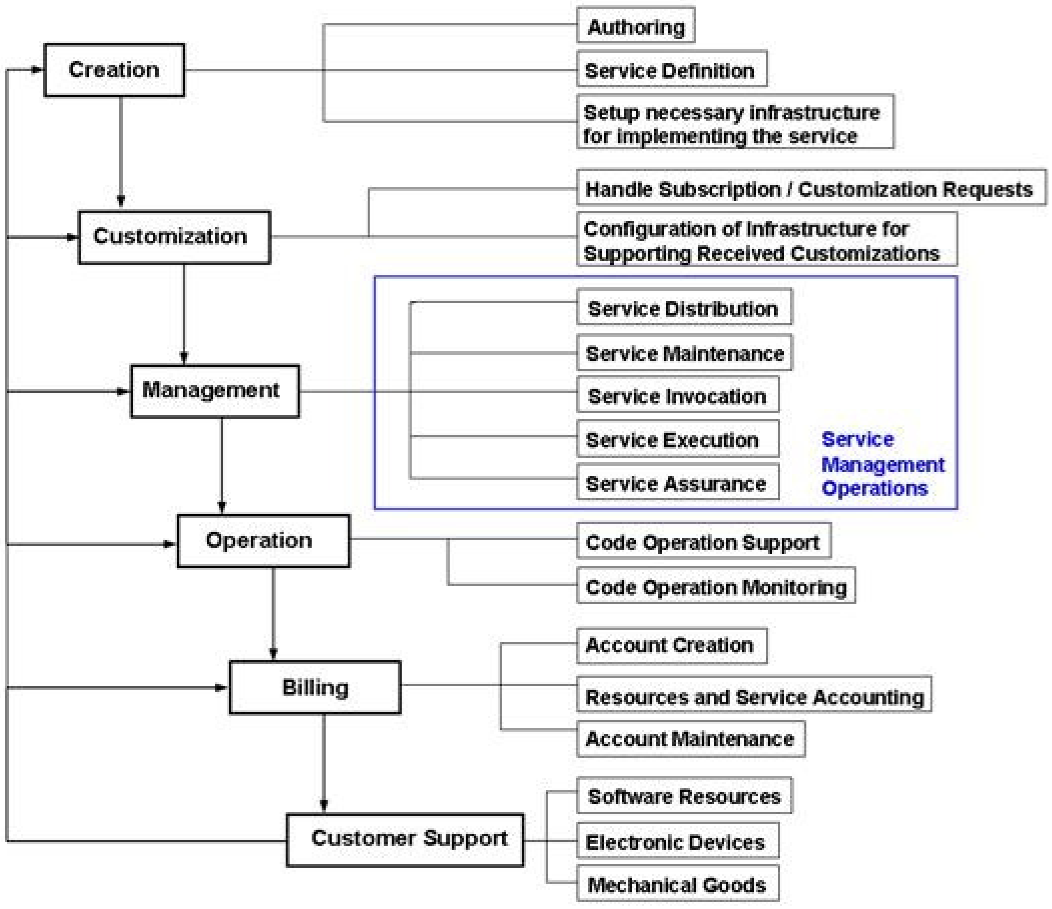


Figure 4-3. Pervasive Service Lifecycle Representation.

Management operations are the core part of the service lifecycle, and are where the contributions of this thesis are focused. Thus, the management phase is highlighted in Figure 4-3. Creation and customisation of services, accounting, billing and customer support are outside the scope of this thesis. The different service phases exposed in this section describe the service lifecycle foundations. The objective is focusing the research efforts in understanding the underlying complexity of service management, as well as to better understand the roles for the technological components that make up the service lifecycle, using interoperable information that is independent of any one specific type of infrastructure that is used in the deployment of NGNs.

4.3.1.1 Service Creation

The creation of each new service starts with a set of requirements; the service at that time exists only as an idea [56]. This idea of the service originates from the requirements produced by market analysis and other business information. At this time, technology-specific resources are not considered in the creation of a service. However, the infrastructure for provisioning this service must be abstracted in order to implement the business-facing aspects of the service as specified in a service definition process.

The idea of a service must be translated into a technical description of a new service, encompassing all the necessary functionality for fulfilling the requirements of that service. A service is conceptualised as the instructions or set of instructions to provide the necessary mechanism to provide the service itself and called service logic (SLO).

4.3.1.2 Service Customisation

Service customisation, which is also called authoring, is necessary to enable the service provider to offer its consumers the ability to customise aspects of their services according to their personal needs and/or desires. Today, this is a growing trend in web-services and business orientation. An inherent portion of the customisation phase is an extensible infrastructure, which must be able to handle service subscription/customisation requests from administrators as well as consumers.

4.3.1.3 Service Management

The main service management tasks are service distribution, service maintenance, service invocation, service execution and service assurance. These tasks are described in Section 4.3.2.

4.3.1.4 Service Operation

The operation of a deployed service is based on monitoring aspects of the network(s) that support that service, and variables that can modify the features and/or perceived status of the communications. Usually, monitoring tasks are done using agents, as they are extensible, can accommodate a wide variety of information, and are easy to deploy. The information is processed by the agent and/or by middleware that can translate raw data into a form having explicit semantics that suit the needs of different applications.

4.3.1.5 Service Billing

Service billing is just as important as service management, since without the ability to bill for delivered services, the organisation providing those services cannot make money. Service billing is often based on using one or more accounting mechanisms that charge the customer based on the resources used in the network. In the billing phase, the information required varies during the business lifecycle, and may require additional resources to support the billing. The make up of different billing infrastructures is out of scope of this work. However, since the management of service information is just as important as the execution and maintenance of the service, this work defines information and processes that can be used for both purposes.

4.3.1.6 Customer Support

Customer support provides assistance with purchased services, computational resources or software, or other support goods. Therefore, a range of services and resources related are required to facilitate the maintenance and operation of the services, and additional context (and sometimes the uncovering of implicit semantics) is necessary in order for user or operators to understand problems with purchased services and resources. A basic business requirement is that the capability for an easy and rapid introduction of new services is mandatory.

4.3.2 Interactions in Service Management Operations

To support the interactions between organisations in a pervasive service, a logic allowing decision-making and the subsequent choosing of alternatives as result must be used. The use of policy-based paradigm in conjunction with ontologies allows this, policies are used to control which parts of the ontology are managed, and ontologies can be used to represent the different semantics that are operated on using logic functions. Policies by themselves usually have no logic operations, but policy-based management using ontologies allows this semantic control.

Use cases are an excellent way to define the interaction between stakeholders in the system, and can be used to simplify and better understand the activity in complex scenarios. Semantic functions are enabled using the concepts that build up later the ontologies. By now in this section, both UML use cases as well as other approaches are used to extract the meaning of the use case(s) and links, and hence provide meaning to the system representations. The basic management operations to support the lifecycle of pervasive services will be described.

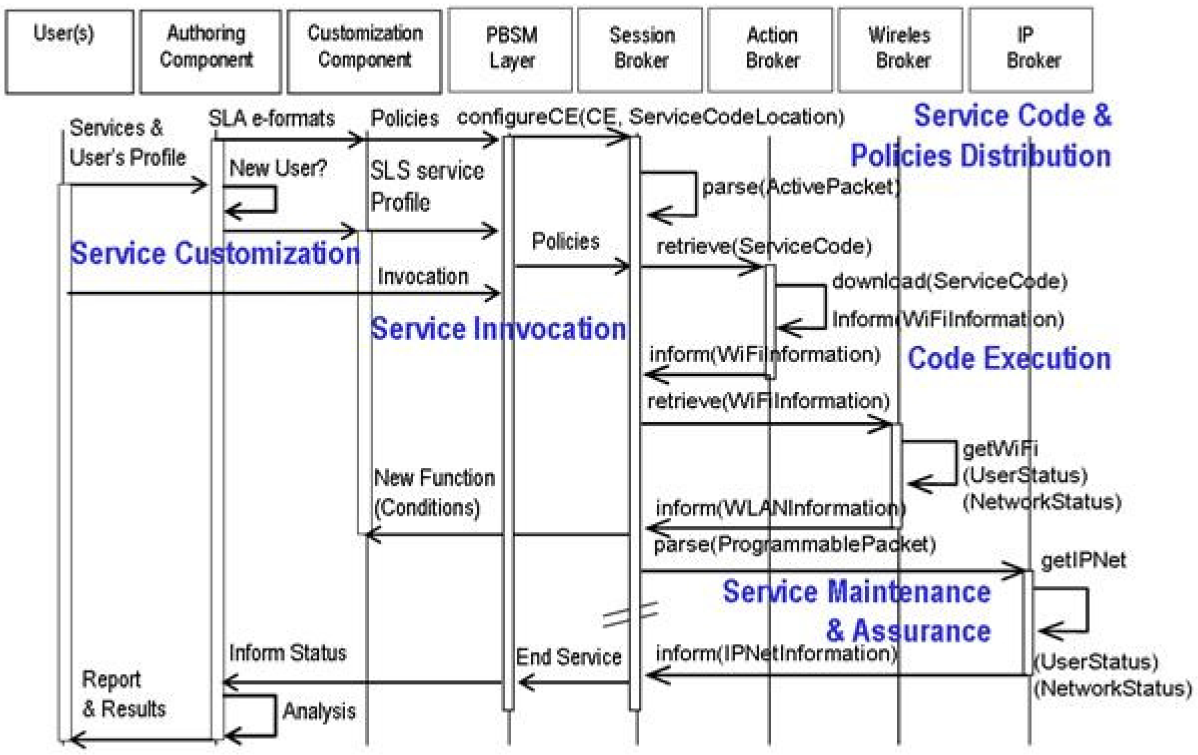


Figure 4-4. Representation for Service Management Operations – Sequence Diagram.

Another tool for representing service management operations that is able to identify the associated semantic interactions is the use of sequence diagrams to represent the variations and conditions in the service [57], as is shown in Figure 4-4. This shows a sequence diagram for pervasive services, where the common operations required to manage pervasive services are described as a communication process between two actors in the service lifecycle. The basic operations are highlighted as common operations present in most pervasive services.

Requirements dictated by pervasive services in terms of information [58] use different types of models that have been proposed in order to enhance information interoperability [58] [59]. However, none of these requirements have been related with service functions or management operations. Knowing the information requirements dictated by pervasive services and the different types of models that have been proposed so far for information interoperability, a relationship between those models and management functions must be established. The purpose of this relationship is to quantify how much each relationship supports or contributes to controlling the service lifecycle phases supporting the management operations.

4.3.2.1 *Service Distribution*

This step takes place immediately after the service creation and customisation in the service lifecycle. It consists of storing the service code in specific storage points. Policies controlling this phase are termed code distribution policies (*Distribution*). The mechanism controlling the code distribution will determine the specific set of storage points that the code should be stored in. The enforcement will be carried out by the components that are typically called Code Distribution Action Consumers.

4.3.2.2 *Service Maintenance*

Once the code is distributed, it must be maintained in order to support updates and new versions. For this task, we use special policies, termed code maintenance Policies (*CMaintenance*). These policies control the maintenance activities carried out by the system on the code of specific services. A typical trigger for these policies could be the creation of a new code version or the usage of a service by the consumer. The actions include code removal, update and redistribution. These policies will be enforced by the component that is typically named the Code Distribution Action Consumer.

4.3.2.3 *Service Invocation*

The service invocation is controlled by special policies that are called *SInvocation* Policies. The service invocation tasks are realised by components named Condition Evaluators, which detect specific triggers produced by the service consumers. These triggers also contain the necessary information that policies require in order to determine the associated actions. These actions will consist of addressing a specific code repository and sending the code to specific execution environments in the network. The policy enforcement takes place in the Code Execution Controller Action Consumer.

4.3.2.4 *Service Execution*

Code execution policies, named *CExecution* policies, will govern how the service code is executed. This means that the decision about where to execute the service code is based on one or more factors (e.g., using performance data monitored from different network nodes, or based on one or more context parameters, such as location or user identity). The typical components with the capability to execute these activities are commonly named Service Assurance Action Consumers, which evaluate network conditions. Enforcement of these policies will be the responsibility of the components that are typically called Code Execution Controller Action Consumers.

4.3.2.5 *Service Assurance*

This phase is under the control of special policies termed service assurance policies, termed *SAssurance*, which are intended to specify the system behaviour under service quality violations. Rule conditions are evaluated by the Service Assurance Condition Evaluator. These policies include preventive or proactive actions, which will be enforced by the component typically called the Service Assurance Action Consumer. Information consistency and completeness is guaranteed by a policy-driven system, which is assumed to reside in the service creation and customisation framework.

Specifically, in this phase the externally provided information can either match pre-defined schema elements to achieve certain management activities or, more importantly, the management systems can use these schema elements to extend and share the information to other management systems. This extension requires machine-based reasoning to determine the semantics and relationships between the new data and the previously modelled data.

4.4 Ontology for Support and Management (OSM)

OSM is a global domain ontology that captures the consensual knowledge about context information, and includes a vocabulary of terms with a precise and formal specification of their associated meanings that can be used in heterogeneous information systems. The Ontology is founded on the diverse information models presented and described previously, including those for context information, policy information, and management operations. The OSM ontology can be seen as the integration of diverse terms, and provides the ability to relate the semantics of separate terms defined in different models to each other using ontology-based formal mechanisms. This is mandatory in order to achieve the interoperability for sharing information in current and especially future communications systems where context, policies and ontologies support such dynamism.

OSM assists human operators by creating a formal mechanism that enables different computing systems to share and reuse different types of information. Most systems are designed using vendor- and application-specific data, which creates multiple applications that are unable to share (what should be) common management data. These so called “stovepipe applications” greatly restrict the sharing and reusing of valuable management data, making network management even more complex. Hence, the way to achieve the aforementioned information interoperability is by introducing a novel semantic plane [60].

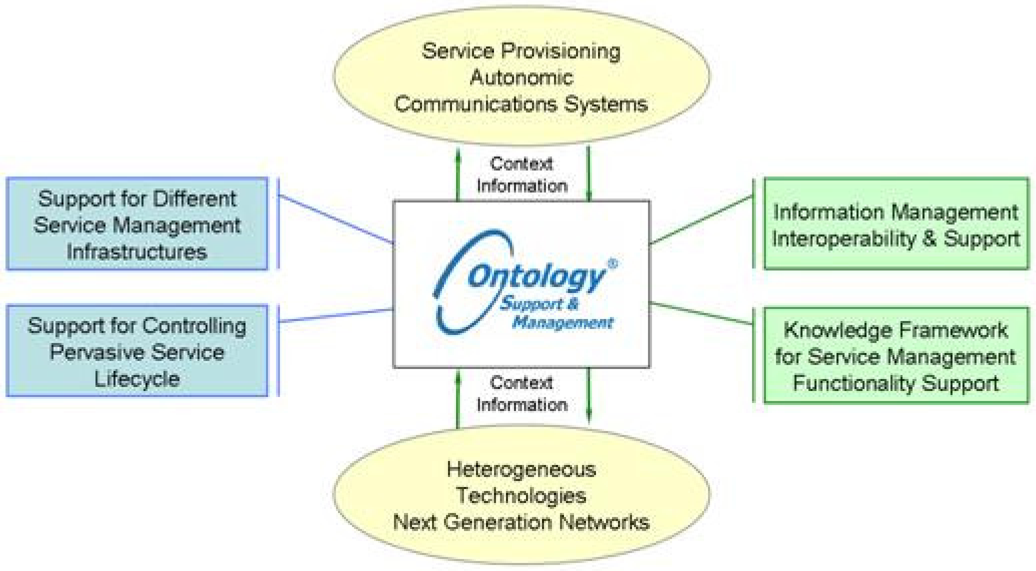


Figure 4-5. OSM – Ontology Support and Management Scope.

The scope of OSM is summarised in Figure 4-5. OSM facilitates the efficient middleware interaction between different operational and business support systems (blue boxes) to create architectures with a common information model that support information exchange that are inherently reusable and shareable. In most communication systems information exchange could be supported by the high-level semantics embedded in management data by means of ontologies [61]. The multiple interactions for information exchange and/or management between context-aware systems, rule-based service systems and policy-driven management systems promotes the definition of new extensible and scalable semantic frameworks for the integration of context information models with ontologies.

OSM (centre of figure) is driven by the set of use cases which focus on service lifecycle management operations using a policy-based architecture for the deployment and provisioning of services. OSM is the result of identified weaknesses in the semantic expressiveness when services are being deployed using existing model-based management architectures.

4.4.1 Purpose and Scope Definition

The first activity to build the OSM ontology involves identifying the purpose and scope that the ontology will have, which depends on the application(s) that will use the ontologies and, more importantly, the extensions required to facilitate the sharing and reuse of knowledge. The hypothesis in this approach acts as a guideline for determining the scope of the ontology. This hypothesis proposes the use of ontologies to formalise the terminology related to the domain of pervasive management. Context information is used to solve one of the main problems in the integrated management - the complexity for managing the associated pervasive service lifecycle operations (creation, customisation, delivery, execution and maintenance of pervasive services) in heterogeneous and cross-layer environments.

4.4.1.1 General Scenario

A realistic autonomic scenario is shown in figure 5-13. This scenario is based on the use of different forms of context as well as the convergence of diverse technologies and heterogeneous networks. The context information supports different tasks in different domains for the creation, customisation, deployment, and most importantly, the management of services. In the depicted autonomic environment, context information is translated into a common form and propagated from each knowledge source, enabling this context information to be available to different management systems that make up the autonomic environment.

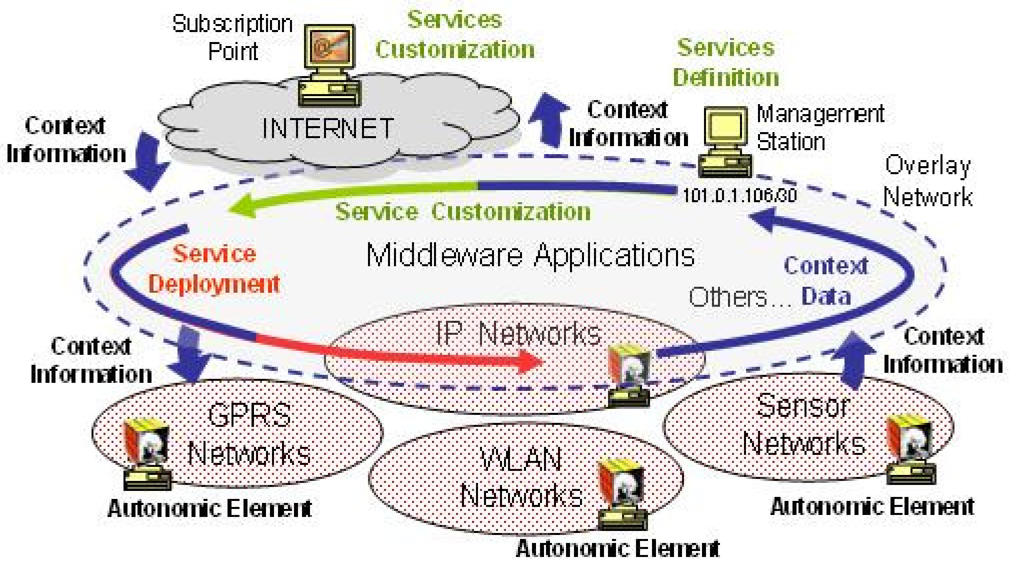


Figure 4-6. Context Information Role in Autonomic Environments.

In the scenario shown in Figure 4-6, ontologies represent and formalise domain-specific context information, so that a set of applicable policies can be used to govern the system management operations. This is done by having context select applicable policies, which are then used to define which roles managed elements (e.g., people, resources, services, and even locations.) are assigned to support the required service [62] [63]. The roles in turn define the characteristics, behaviour, and/or functionality that can be used by the management applications to create, change, manage, and remove requested services. We build upon and extend the scenario for integrating context in pervasive services from the EU IST-CONTEXT project, using ontologies as a formal mechanism to integrate both the context information and the policy-based services in the same management system.

4.4.1.2 Scenario Representation

The process of building an ontology requires a set of tools, due to its inherent complexity and many relationships. A simple graphical representation is an optional, but helpful, aid in this process. This is especially important for engaging the business constituency, which is not comfortable with UML and similar approaches. Note the emphasis on simplicity, which can be used for multiple scenarios, rather than complex mechanisms.

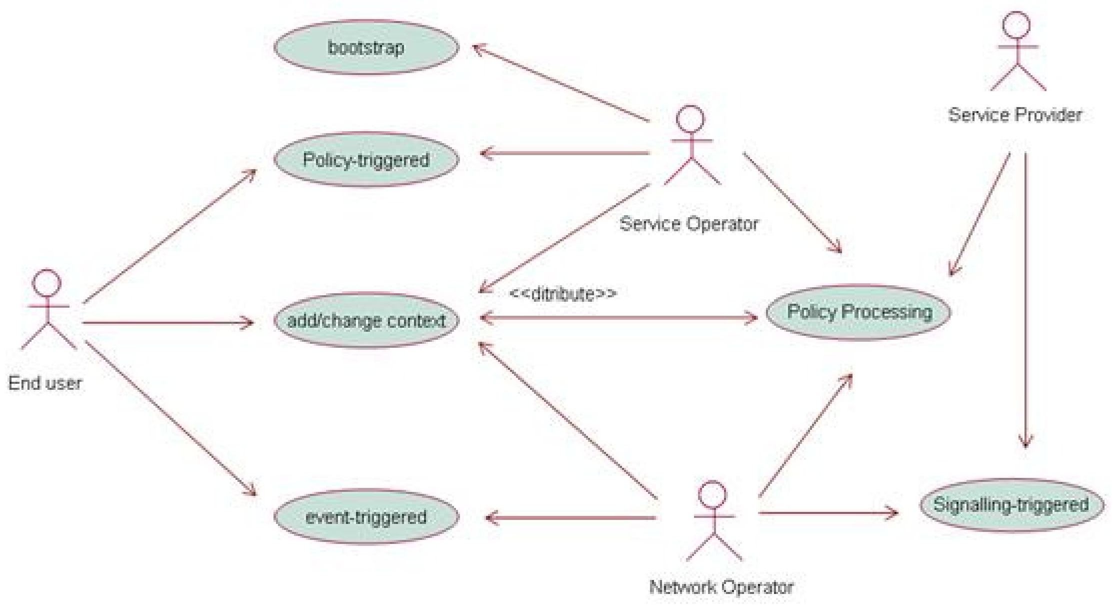


Figure 4-7. OSM - Service Lifecycle Operations Use Case Diagram.

Figure 4-7 shows the use case representation for pervasive services, where the common operations required to manage the services, which are present in the above scenarios but are often difficult to visualise, are described as processes between two actors. The basic operations are highlighted as common operations that are present in most of the pervasive services. Use cases are an excellent way to define the interaction between stakeholders in the system and, as this thesis shows, should always be used to simplify and better understand the activity and interactions between different processes in the scenarios. Experiences with both UML use cases as well as sequence diagrams are used to better understand the scenario(s). Use cases and sequence diagrams are used to extract the meaning of the scenario links, provide meaning to the system representations.

4.4.1.3 Specialisation

Specialisation is an inherent property of the ontology. This activity depends on the concepts present in the domain that the ontology will support, rather than the number of concepts that the ontology will have.

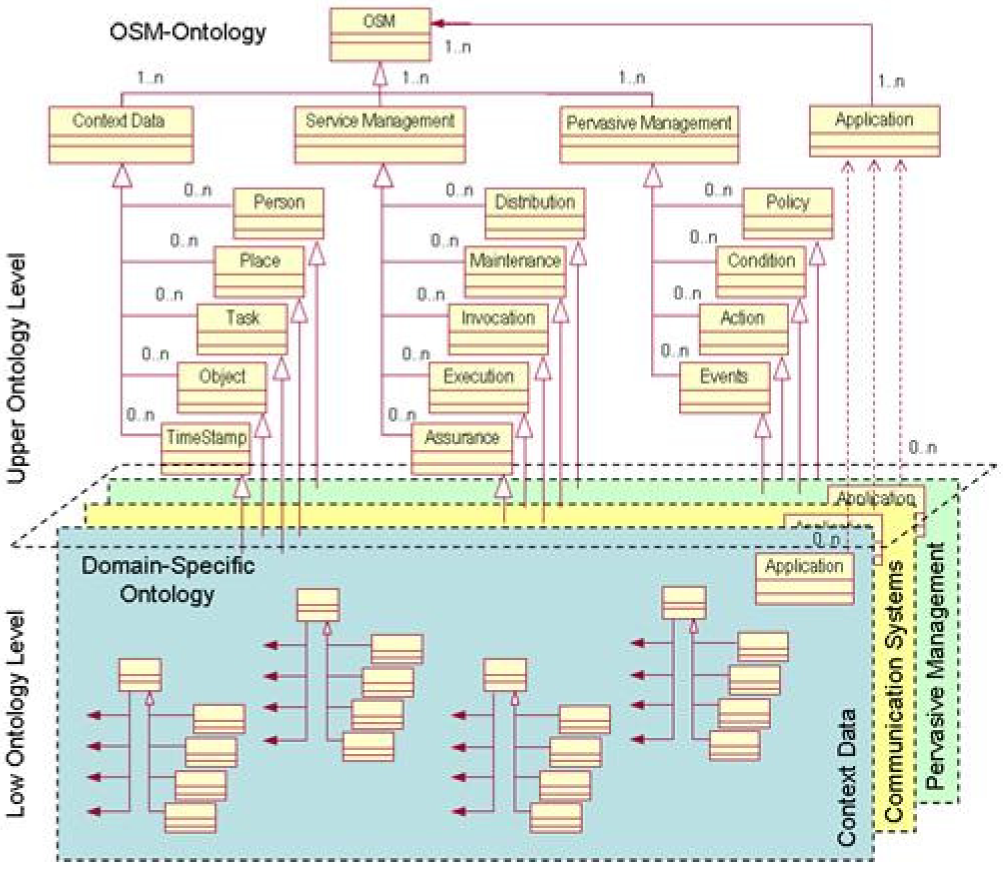


Figure 4-8. OSM - Upper Ontology Representation.

Specialisation enables concepts from different knowledge domains to be combined. This is especially important for pervasive applications and autonomic communications, which demand the integration of highly specialised domain information, such as networking, management principles, organisations of people and devices, and roles. Therefore, the level of specialisation is high; this enables the ontology to accurately model the management and context concepts, and hence more precisely model management operations. This work built an "upper ontology" to standardise concepts that are generic to different knowledge domains, and a set of "lower", or domain-specific, ontologies to represent vendor-, network-, technology-, and domain-specific concepts. Figure 4-8 shows the representation of the upper ontology proposed here along with the low-level ontologies that are used, and shows how the low-level ontologies are related to the upper ontology. OSM was built in an iterative manner, where some concepts from the upper ontology were used to guide the development of domain-specific concepts, and vice-versa.

4.4.1.4 Terms Definition

The definition of specialised terms is the formal start of most ontology development activities. An informal list of concepts identified from the scenarios can be useful to guide the overall structure of the ontology, as well as a list of nouns and verbs (corresponding to classes and attributes, respectively) from the information models. However, these latter two approaches are most useful to guide the development of concepts and relationships using a formal definition and language. Furthermore, informal lists of concepts are useful at the beginning of development to help construct the ontology, but each list must be periodically reviewed and refined in order to avoid ambiguity and future misunderstandings.

OSM defines a set of descriptions for the three abstractions levels (context data, pervasive management, and communication systems) as part of each related domain. The “definitions” use natural language, and are included in the OSM ontology constructions. Additional semantic rules are added to construct a formal lexicon. The formal descriptions about the terminology related to context information, pervasive management and communications networks domains are included to build and enrich the proposal. Thus, the OSM ontology solves one of the main problems in the management of services and networks: the integration of context information into tasks for managing service operations.

4.4.2 The Formal Definition of the OSM Ontology

Most ontology-based proposals do not address the relationship between the different domains of context data, policy models and communication networks and its application and usage as studied in [64]. This section describes the guidelines for a novel approach, whose specific aim is to integrate ontological data with information and data models, thereby creating a powerful semantic representation of formal knowledge oriented for supporting pervasive service and network management operations. While there is no one correct way to model a domain, let alone build an ontology, there must be a set of common, tested steps that can address our ontology development activity in order to share and reuse its concepts [65].

4.4.2.1 OSM Interactions

The OSM interactions using ontologies are represented as the relationships between the classes from the different ontology-based model representations. Thus, the context information model, the policy information model, and the service management operations, as shown in Figure 4-9, all have different relationships that represent these class interactions. The ontology-based models contain class correspondences, and the interactions are used to construct OSM interactions which are shown in the central part of the figure.

The objective of defining these class interactions is to identify the classes whose content must be integrated with all or part of the context information. The task of identifying these interactions is done with a visual approach using the ontology class diagrams based on the individual information model class definitions that have been previously presented and explained. The class diagrams contain the class relationships described in a generic form (e.g., as abstract classes interacting for deploying a service), but each relationship and interaction between classes is appropriately refined when the ontology is constructed and subsequently edited. Thus, the class interactions map becomes a tool for representing the semantic interactions which make up the ontology.

It is important to highlight the difference between interaction and relationship. Interaction is the graphical representation for two or more elements in class diagrams that interact and hence define one or more specific behaviours. This is normally shown using symbols that represent the type of interaction, so that different interactions that have different meanings can be easily identified. On the other hand, relationship is the semantic description of the interaction between two or more elements, usually consisting of a description that follows simple syntax rules and uses specific keywords (e.g., "*isUsedFor*" or "*asPartOf*").

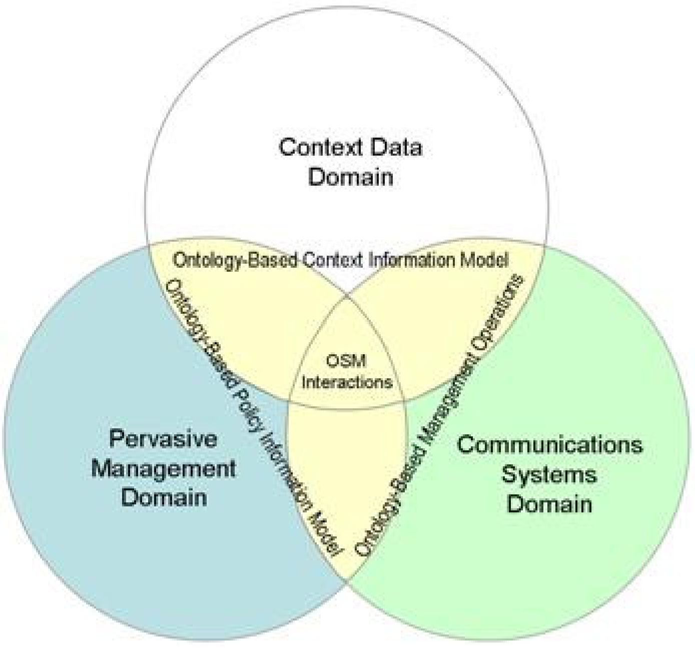

Context Data Domain
Ontology-Based Context Information Model
Ontology-Based Policy Information Model
OSM Interactions
Ontology-Based Management Operations
Pervasive Management Domain
Communications Systems Domain

Figure 4-9. OSM – Upper Ontology Representation.

Figure 4-10 shows the upper ontology of the OSM ontology (OSM Upper-representation). The image represents the integration of the context information classes related to the management operation class through the event class.

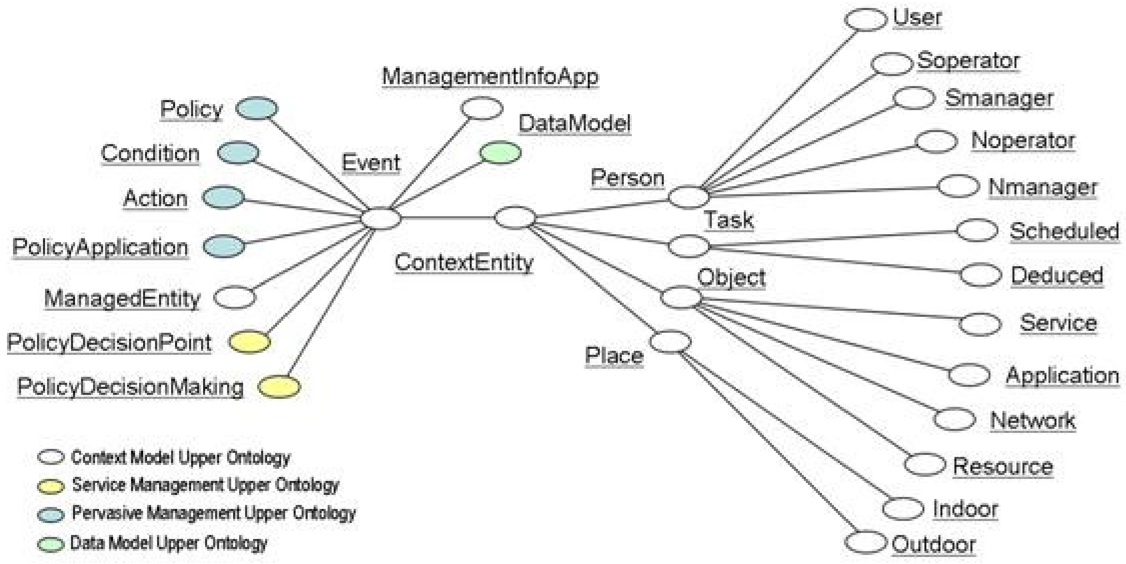


Figure 4-10. OSM – Representation of the Integration of Context in Pervasive Management.

The Event class interacts with other classes from different domains in order to represent context information. Note that only the ContextEntity class from the context information model domain and the Event class from the service management domain are shown. This simplifies the identification of interactions between these information models. These entity concepts, and their mutual relationships, are represented as lines in the figure. For instance, a ContextEntity forms part of an Event class, and then the Event defines part of the requirement evaluating a Condition as true or false. Another example shows that one or more Policies govern the functionality of a Managed Entity by taking into account context information contained in Events. This functionality to enables context to change the operation requested from a pervasive service or application, and is represented as the interaction between Event and ContextEntity.

The OSM ontology integrates concepts from the IETF policy standards as well as the current DEN-ng model. Thus, in the following subsections, important classes that were originally defined in the IETF, DEN-ng, or OSM models will be identified as such.

The representation of the OSM ontology is based on supperclass interactions between the classes from the domains involved in the integration of context information for pervasive service management operations. Figure 4-11 shows a simplified view of such interactions for some of the OSM Upper Ontology classes.

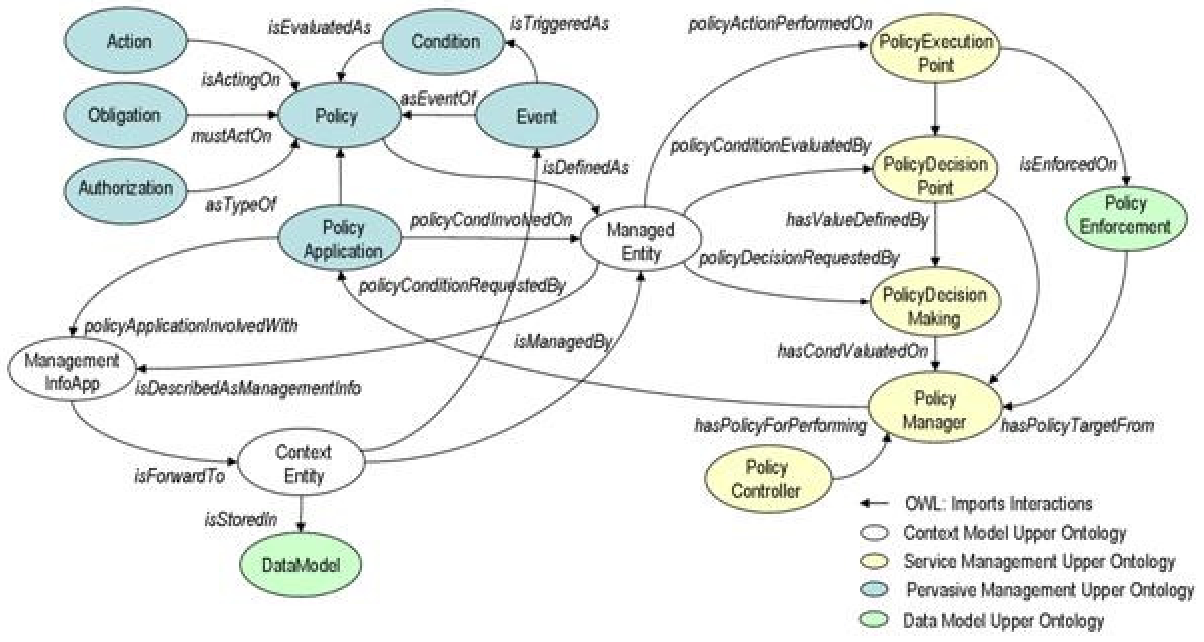


Figure 4-11. Representation of Upper Class Interactions in OSM Ontology.

From the DEN-ng model, ManagedEntity is the superclass for Products, Resources and Services, and it defines the business as well as the technical characteristics for each. A DEN-ng PolicyApplication is a type of Application. It evaluates a set of PolicyConditions and, based on the result of the evaluation, chooses one or more PolicyActions to execute. A PolicyController is the DEN-ng equivalent of a “Policy Server” from other models; the name was changed to emphasise the fact that PolicyApplications are not limited in implementation to the traditional client-server model. The OSM class interaction *hasPolicyForPerforming* signals when the set of OSM PolicyRules, selected by the current OSM ContextData, are ready to be executed. When this interaction is instantiated, the PolicyController can pass the set of policies to the PolicyManager, which is an OSM class.

A PolicyManager is a fundamental class, and represents both a set of core functionality for implementing policies as well as a unit of distribution in a distributed implementation. The PolicyManager works in coordination with a set of specialised policy components, including the PolicyExecutionPoint, PolicyDecisionPoint, and PolicyDecisionMaking classes (all defined by OSM); other components are not shown in this figure for the sake of simplicity. These specialised policy components enable application-specific functionality of a PolicyManager to be built based around the use of a set of particular policy-based management operations, many of which can be defined as reusable components.

An OSM PolicyDecisionPoint (PDP) is similar to traditional PDPs (such as that defined in the IETF), except that it is specifically designed to answer requests from policy-aware and policy-enabled network elements (as represented by the ManagedEntity class interaction *policyConditionEvaluatedBy*) using formal ontology-based terms. This enables a PDP to serve as an interface between the network and higher-level business processes. The difference between a policy-aware and a policy-enabled entity is a longer discussion that is beyond the scope of this work. For now, it is enough to say that the semantic expressiveness reached when using the combination of policy-aware entities and ontology-based mechanisms for representing and integrating context makes this proposal different from traditional policy-enabled approaches.

The OSM PolicyExecutionPoint is used to execute a prescribed set of policy actions on a given ManagedEntity as the class interaction *policyActionsPerformedOn*. A PolicyEnforcementPoint is a class that performs the action on a given ManagedEntity using the class interaction *isEnforcedOn*.

The PolicyDecisionMaking entity received requests from a ManagedEntity and evaluates PolicyConditions (via the class interaction *hasCondValuatedOn*) based in part from the results of the PDP (as a result of the class interaction *hasValueDefinedBy*). The class interaction *policyConditionRequestedBy* tells the PolicyApplication to evaluate one or more specific PolicyConditions; this may depend on retrieving management information using the class interaction isDescribedAsManagementInfo (which involves the ManagementInfoApp and the ManagedEntity). The class interaction *isForwardTo* is established when it is necessary to operate on the ContextEntity to store the context information in the DataModel using the class interaction *isStoredIn*.

In order to integrate context information properly, the *isDefinedAs* class interaction (between ContextEntity and Event) enables the Event to trigger the evaluation of a PolicyCondition, as it is part of the Condition values of the policy-aware information model. This is defined as the two class interactions *isTriggeredAs* and *isEvaluatedAs* (with the Condition and Policy components, respectively). The PolicyDecisionPoint obtains the status of the ManagedEntity after the execution of the PolicyAction(s) using the *policyApplicationInvovedWith* and the *isDescribedAsManagementInfo* class interactions. The class interaction *hasPolicyTargetFrom* is used to check if the PolicyActions have correctly executed.

4.4.2.2 Capture

The capture of the ontology refers to the knowledge acquisition task, which progresses from the conceptual descriptions to the definition of terms in a natural language, and continues with a formal representation of the acquired knowledge. As it is described along of this chapter the formal language used to represent the ontology is OWL. However for implementation factors and particularly in this work, the use of XML in the above stage (conceptual descriptions and its representation) has been demonstrated and justified, as it provides implementation advantages when the java programming language and java technologies are being used; the advantages are as listed below:

- XML is a mark-up language for documents containing structured information, but can also be used as a mechanism to exchange and store data;

- XML can be used as a data storage and exchange mechanism;
- XML documents can be automatically validated using DTDs (Document Type Definitions) or XSDs (XML Schema Definitions) when they are created and/or modified (in this thesis, the implementation has been validated using a JAVA program, which is the same program used for creating and maintaining XML docs);
- Searching is made efficient through the use of XQuery, XPath, and other XML-based tools.

4.4.2.3 Coding

This section describes the concepts defined and represented for building the OSM ontology, and describes how the ontology is formalised in terms of code generation. The structuring of the knowledge (concepts from the real world) in an ontology-based model pertaining to a virtual and a particular domain can be performed using one or more parsers to interface between the specific semantic concepts and their descriptions in two or more domains. Once the semantic concepts are coded, a formal language is generated for machine usage.

Currently, the biggest ontology driver is the Semantic Web. Software tools are available to accomplish most aspects of ontology development. While ontology editors are useful during each step outlined above, other types of ontology building tools are also needed along the way. Ontology languages increasingly rely on W3C technologies. With the development of the Semantic Web, a new family of languages appeared, but most of these are based on the Resource Description Framework (RDF) as result of its easy adaptability and extensibility.

The development tool chosen to edit the OSM ontology is an open source solution (Protégé2000). This tool can be used to construct ontologies, providing the ability to customise data entry forms and enter data. It can also be used to easily include graphical components (graphs, tables). In addition, it uses various storage formats such as OWL, RDF, XML, and HTML, which enable its results to be easily integrated with other system components. Protégé uses the concept of plug-ins to add and integrate new functionality. Three types of plug-ins exist:

- Slot widgets, which provide custom data creation, editing, deleting, and management capabilities including formatting.
- Tab plug-ins, which provide significant added functionality available under a new tab in the ontology editor.
- Backends, which provide import, export, storage and retrieval facilities.

4.5 Conclusions

The complex task of brokering context information between all involved actors has been discussed in this section. Based on the description of a middleware and an ontology-based information model, the concrete integration of both has been described in great detail. Due to the combination of relatively new technologies it has to be dealt with unexpected challenges but the approach shows the feasibility and with proceeding development, the advantages will be even more substantial.

Though this conceptual change in the CoCo Infrastructure works quite well there remains room for improvement. At the moment, context information is stored in the database but never erased or moved. While historical context information surely is useful, databases will get out of hand without a fitting algorithm to clear the database possibly relying on the Quality of Context. Secondly, with the improvement of OWL databases and inference engines a lot more efficient solutions will be possible.

This work has also proposed the use of functions following set theory and semantic rules for controlling the management operations of the service lifecycle. The solution seems suitable for any particular technology, as it is following self-management principles inspired from autonomic communications. The comparative results between a management system using policies with and without semantic enrichment and ontology-based functions illustrate the differences of using this approach. Our approach is a formal alternative to represent the service lifecycle operations. We concentrate on representing service management operations for supporting the information interoperability in pervasive environments. Both service and management operations are described and represented with the combination of ontologies and information models.

We presented the ontology for supporting the creation, delivery and maintenance of pervasive services, and also for the integration of the user's context information in service management operations, OSM (Ontology for Support and Management). OSM is developed following a formal methodology in order to provide the necessary semantic richness to represent the different types of information to be integrated in management operations. This has been defined using a formal methodology in order to capture concepts from the context information that is available to help define values used in various service management operations.

OSM is an ontology-based information model for pervasive applications that integrates its information model describing different aspects of context with ontologies that augment the model with semantic information; these augmented data enable context-aware service requirements to be modelled. Structuring this as a formal language using OWL provides extensible and efficient context information for managing services. OSM enables pervasive services to be automatically customised, delivered and managed. OSM is suitable not only for representing and defining context information concepts, but also for defining management operations that motivate and promote the integrated management of pervasive services.

5 Distributed and Autonomic Real-Time Traffic Analysis (DATTA2)

5.1 Introduction

During the last decade, IP networking has constantly grown in popularity. Traditional services that have been deployed over dedicated network infrastructures (such as telephony, radio, or television) have slowly switched to IP. The "everything-over-IP" approach had a direct impact on the network operators, as with each new carried service the amount of carried traffic increased significantly [66]. Such traffic increases highly impact the ability of network operators to perform traffic analysis, which they require for understanding the type of traffic they carry. Traffic profiling within a network operator is particularly important in order to provide better quality services, charge users based on the traffic they generate, plan network upgrades, or detect malicious traffic.

DATTA2 addresses new problems faced by operators from different perspectives presented in the next sections. First, a prototype for coordinated distributed traffic capturing and monitoring (DiCAP) shall be presented. Then, a distributed storage for NetFlow records based on P2P technologies shall be introduced and evaluated.

5.2 DiCAP: Distributed Packet Capturing Architecture for High-Speed Network Links

**A copy of this report has been published in [67].*

One of the key operations in analysis of IP traffic is capturing the traffic from a network link. This proposal differentiates clearly between traffic capturing and traffic analysis and covers the traffic capturing aspect only. IP traffic capturing determines the process of retrieving data from the network link, while traffic analysis defines the process of inspecting that data. Often, in IP traffic analysis (*e.g.* Bro [68] and Snort [69] Intrusion Detection Systems), the analysing device is also responsible for capturing the network traffic. As link bandwidths continuously increased over the last decade at a higher rate compared to CPU (Central Processing Unit) and memory speed, such an approach became unfeasible for high-speed links. Sampling is seen as one solution to leverage this gap as proposed by [70], [71], [72], or [73]. By sampling, a traffic analysis system (such as Intrusion Detection Systems or IP flow exporting device) only looks at a fraction of packets and makes its calculations and decisions based on that set. The main drawback of sampling is the decrease in accuracy of the results in the system using sampling. Other papers, such as [74] and [75], have shown that packet sampling decreases the accuracy of traffic analysis results. In those particular cases it was shown that sampling degrades the quality of the Intrusion Detection Systems (IDS) algorithms which were investigated.

In order to address the problem of sampling, the DATTA project proposes a distributed architecture for IP traffic analysis on high-speed links by using the combined resources of multiple standard PCs. DiCAP represents one component of DATTA responsible for the capture of IP packets at high packet rates.

The main idea of DiCAP is to distribute the analysis work to multiple devices in order to avoid using sampling, or to maximise the number of packets that are sampled, when sampling is used. During early investigations of this work, it was observed that when using a standard PC and standard Linux packet capture libraries, the capturing performance for links with high packet rates was very limited. Therefore, DiCAP was designed in order to cope with high packet rates by distributing the traffic load between several PCs which capture packets in parallel.

One of the main problems of capturing and analysing packets on high-speed links is the very short time that may be spent handling a single packet. As shown in [76], 44% of the IP traffic observed on an Internet Service Provider (ISP) in today's Internet is made of packets with sizes between 40 and 100 bytes. Assuming a 10 Gbps Ethernet link, fully loaded with 64 byte packets, which are very common in voice over IP (VoIP) applications, this would translate into approximately 20 million packets per second, or approximately 50 ns for handling a single packet by the packet capturer. Capturing packets on high speed links requires high-performance memory, typically exceeding the current speed of DRAM (Dynamic Random Access Memory) memory existing in standard PCs. Dedicated capturing cards, such as Endace's DAG cards [77], make use of the faster, more expensive SRAM (Static Random Access Memory) memory, and are able to capture at those high packet rates, but they typically come at much higher prices. DiCAP is primarily designed to achieve high capture rates by using a scalable architecture, which is able to combine the resources of multiple inexpensive, off-the-shelf Linux machines. Additionally, due to its simple design, DiCAP offers a scalable traffic capture solution that could be easily embedded in cheaper hardware capture cards.

5.2.1 Distributed Packet Capturing

Packet capturing performance on high speed network links is highly influenced by the performance of the capturing hardware and software. Two limiting factors for packet capture tools are the memory and CPU speed on the capturing node. The higher the packet rate on the network link, the less time available for capturing and analysis of a single packet. As the evaluation of this solution shows, the capturing performance of a Linux machine drops at high packet rates.

Existing distribution approaches either require dedicated hardware, or are dedicated for a particular protocol, or use sampling. DiCAP proposes a scalable distributed packet capturing architecture based on common, inexpensive hardware. DiCAP is neither designed for a particular protocol, nor is its usage bound to a particular scenario. Its architecture allows it to be used alone, for capturing packet headers at high packet rates, or in combination with other Linux capturing tools, such as *libpcap* or *libpcap-PFRING [78],* in order to increase their capturing performance.

The main idea of DiCAP is to split the packet capturing load among several self-organising capture nodes placed on the network link and have each of them "sample" the traffic in such a way that no two capturing nodes capture the same packet. DiCAP allows for scalable packet capturing by increasing the number of capture nodes, when the increase of traffic requires.

The performance of existing software-based packet capturing tools, such as *libpcap* or *libpcap-PFRING,* decreases at very high packet rates, even on Gigabit Ethernet links, as observed during the evaluation of DiCAP. The performance decrease is due to the processing of each packet by the kernel at several layers. By parallelising the packet capture, the number of packets captured by a single node is decreased, thus, the time available to process a single packet is higher and chances to drop packets due to lack of resources is smaller.

The DiCAP architecture (Figure 5-1) is made of several capture nodes, which are organised within a capture cluster. The *capture nodes* are coordinated by a node coordinator. The *node coordinator* controls the capturing process by configuring the rules that each *capture node* has to follow during the capture process. The architecture shows a single logical component for the coordination task, but, for increased robustness, secondary node coordinators could be present, in order to take over the coordination, in case the main coordinator crashes. In case multiple coordinators are used in parallel only a single one actively coordinates the capture nodes.

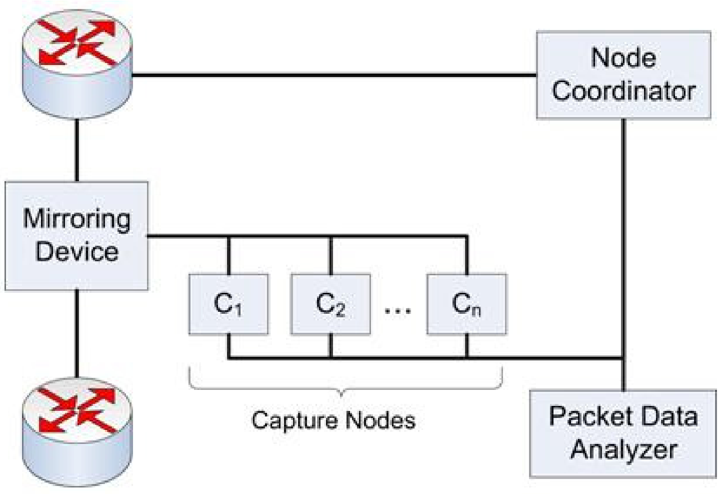


Figure 5-1. DiCAP Architecture.

Moreover, the active *node coordinator* constantly informs the other coordinators about the topology of *capture nodes*. Whenever such a synchronisation message fails to be received by the other coordinators, an election mechanism chooses which of the other coordinators will become the new active *node coordinator.*

The *packet data analyser* may be used to receive capture traffic from the capture nodes and further analyse it. Similarly to the node coordinator, the *packet data analyser* may be distributed. Besides the increase in robustness, distributing the *packet data analyser* can also be used for load balancing. In this case, the *capture nodes* are informed which all the *packet data analysers* are and may randomly select one to which to send the captured data. Live traffic observed on a high-speed network link is mirrored to a capturing cluster which is composed of several capture nodes.

The mirroring device is a central component of DiCAP and its failure may cause the whole network of capture nodes to stop operating. However, the mirroring may be the same device as one of the two routers on the main link, so if that device fails, there will be no traffic to be mirrored anyway. Each of the capture nodes has two network interfaces: a *passive* network interface, on which live traffic is received from the mirrored link, and an *active* interface, that is used to communicate with the *node coordinator(s)* and the *packet data analyser(s)*.

5.2.1.1 Operation

Each capture node in DiCAP has a 32 bit *nodeID.* The *nodeID* is used to uniquely identify the capture node within the cluster of capturing nodes. Once a *nodeID* was generated, the capture node needs to associate with a *node coordinator*. A *capture node* may be preconfigured with details of a *node coordinator*, or those details may be transmitted from time to time by the *node coordinator* using multicast.

During the association process, the *nodeID* is sent to the *node coordinator* in a *join* message. Once the node coordinator receives the *join* message, it adds the corresponding *nodeID* to its list of known active capture nodes. Additionally, the node coordinator informs the whole capturing cluster about the updated cluster topology. The *node coordinator* has the task of informing capture nodes which packets they need to capture, based on one of the two mechanisms described in Section 5.2.1.3.

For keeping a consistent view on the cluster of capture nodes, each *capture node* sends one heartbeat message every 5 seconds to the *node coordinator*. If the *node coordinator* does not receive a heartbeat message for 15 seconds from one of the capture nodes, that *capture node* is assumed offline and it is removed by the *node coordinator* from the list of active capture nodes.

5.2.1.2 Logical Topology of the Capture Cluster

The logical topology of the capture cluster may be seen as a bus and it is created based on a distributed algorithm, which is coordinated by the *node coordinator*. Each *capture node* decides what packets it has to capture based on the rules received from the *node coordinator*. For making this decision, DiCAP provides two options: *round-robin selection* and *hash-based selection*.

In the *round-robin selection* mode*,* capturing nodes are logically organised in a circular list. Each node knows how many active capture nodes are in that circular list and also knows its position in that list. For each incoming packet only a single node is the responsible capture node. That node captures the packet, while the next node in the circular list becomes the new responsible capture node.

In *hash-based selection,* the capturing nodes form a Distributed Hash Table (DHT), so that each node is responsible with a particular range of hash values. For every incoming packet, each capturing node applies a deterministic hash function on a subset of the packet header fields and captures the packet if the resulting value is in its range of responsibility. A more detailed explanation of the two selection mechanisms is given in Section 5.2.1.3.

The packet selection algorithm highly impacts the way the capture load is distributed among the *capture nodes.* The *round-robin selection* allows for a perfect distribution of workload, but it requires injection of control packets by the *node coordinator* into the monitored traffic. The *hash-based selection* can be totally passive with respect to the monitored traffic, but it has two disadvantages:

- A hash function which equally distributes traffic in any condition is difficult to find.
- Calculating a hash for each packet is more computational-intensive than just increasing a counter (as in *round-robin selection*).

The type of selection to be used, as well as other management information, is communicated to *capture nodes* by the *node coordinator* using so-called control messages (Figure 5-2). Based on different message types exchanged between *capture nodes* and the *node coordinator,* a generic control message format has been defined.

Each control message contains a type (TYP) field, which specifies the control message type. Three types of control messages have been defined for DiCAP: *join, accept, and topology update*. The information in the control message is organised in AVPs (Attribute-Value-Pairs). Just after the type field of the control message, the length (LEN) field specifies how many AVPs are contained within the control message. Figure 5-2 also shows the format of an AVP. An 8 bit code specifies the attribute type and it is followed by a 32 bit value for that attribute.

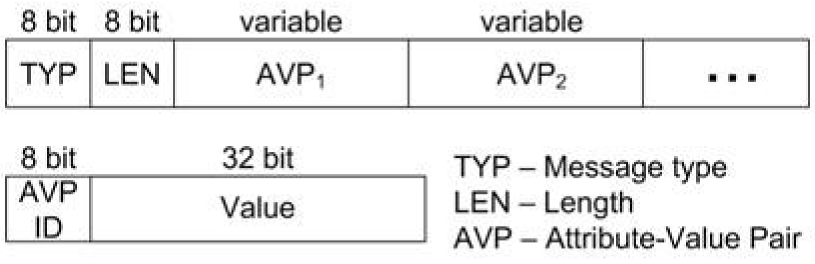


Figure 5-2. Control Message.

Figure 5-3 shows the different type of AVPs defined for DiCAP. The *Node ID* AVP is used in all three control messages. In the join message, *Node ID* specifies the *capture node* identifier for a new node that wants to join the *capture cluster*. In the accept message, *Node ID* specifies the intended capture node for which the accept is issued. In the *topology update* message, a list of *Node ID*s is used to define the topology of the capture cluster. The *Coordinator IP* and *Coordinator Port* AVPs are used to inform the *capture nodes* about which IP address and port number the synchronisation messages will use as a source. The *Analyser IP* and *Analyser Port* specify the IP address and the port number where captured data should be relayed to for further analysis. The *Selection Type* AVP specifies which type of selection the *capture nodes* should use (*round robin* or *hash-based selection*).

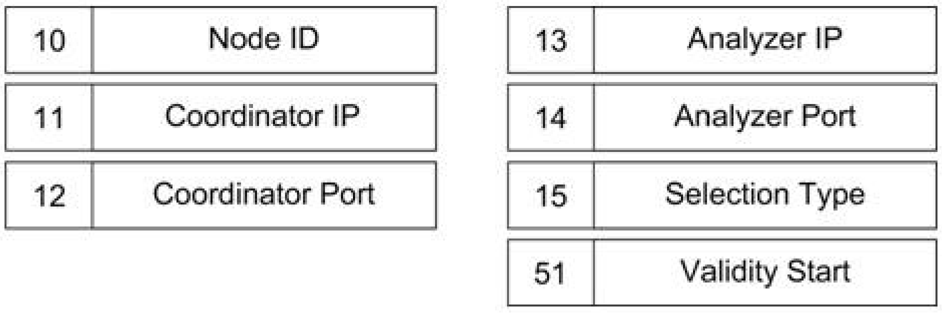

10 Node ID
11 Coordinator IP
12 Coordinator Port
13 Analyzer IP
14 Analyzer Port
15 Selection Type
51 Validity Start

Figure 5-3. DiCAP AVPs.

5.2.1.3 Synchronisation

The *node coordinator* maintains a list of *nodeIDs* for the active *capture nodes*. Each of those *nodeIDs* is kept in the list for as long as the *node coordinator* receives heartbeat messages from the respective *capture node*. Regular updates are sent by the *node coordinator* to all *capture nodes* to inform them about the topology of the *capture cluster*. Based on the topology and the selection method, each node then decides by itself which packets it has to capture.

In the *round-robin selection*, the topology update control messages are sent in-line, as packets injected in the monitored traffic. Topology update messages may be sent every *N* seconds, or, whenever the *node coordinator* learns about a new active *capture node*, or detects that a *capture node* is down. The update message contains a list with the *node IDs* of all active *capture nodes*. Upon the receipt of such an update message, each capture node knows about the total number of *capture nodes* (*N)* and it's own current position *(P_i)* in the list of *capture nodes*. A packet counter *C* is reset to 0 by the receipt of a topology update control packet. The packet counter is incremented with each packet seen on the wire. A capturer captures a packet if, after the receipt of the packet, the following equation holds true: $C \bmod N = P_i$

The round-robin capture mechanism is also depicted in Figure 5-4. The figure assumes that the *capture node* is the third node in the list of capturers. Once it receives the topology update message, it resets the counter *C* to *0*. The figure shows that the first two packets are dropped, while the third one is captured. The counter shall be again set to *0* upon the receipt of the next topology update message.

In order for this mechanism to work, it is required that each capturer receives all packets in the same order. The mirroring device needs to ensure that the traffic is sent in the same order on all ports which forward the mirrored data. If such a system that assures that all capturers receive the packets in the same order cannot be satisfied, the decision to capture a packet needs to be taken based on a deterministic hash function which is applied to a subset of the header fields of each IP packet. In this case, each *capture node* captures the packet if and only if the result of the applied hash function is a particular value for which the node is responsible.

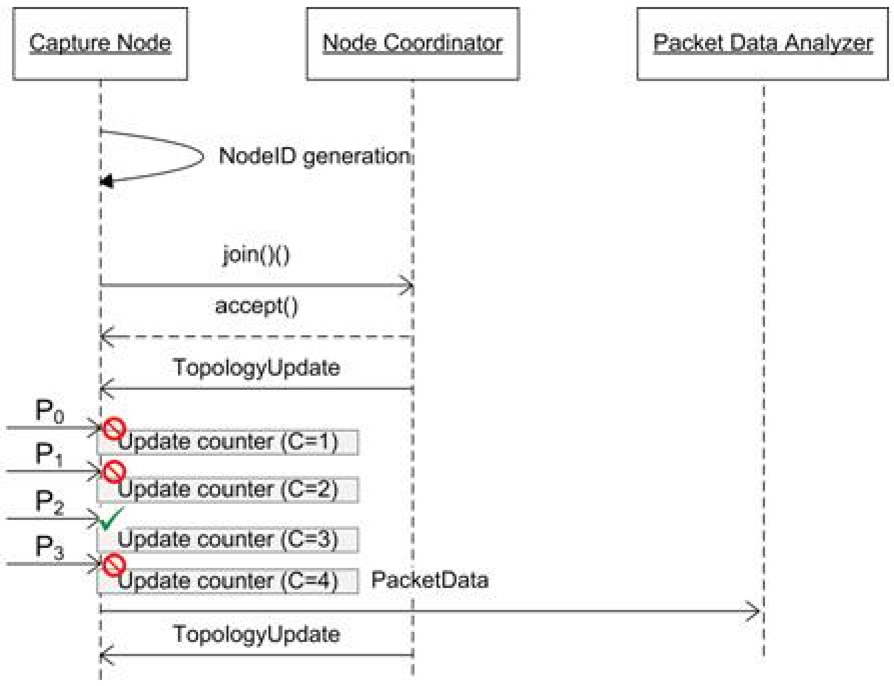


Figure 5-4. DiCAP Communication.

As the hash values determine which node captures a particular packet, in order to fairly distribute the workload between the different *capture nodes* in this second approach, the hash values generated by the chosen hash function need to follow as close as possible a uniform distribution function. One obvious choice would be to apply the hash function on the IP address and port number fields in the IP header, but that has the big disadvantage that in situations in which an increased amount of traffic flows to/from the same IP address/port number, such as large data flows or denial-of-service attacks, the node responsible for the respective IP address or port number gets overloaded. In order to avoid such situations, DiCAP uses a hash function which is applied to the identifier field of the IP packet header. The identification field of each IP packet contains a 16 bit value that is used to identify the fragments of one datagram from those of another. The IP protocol specification require~~s~~ that the value must be unique for each source-destination pair and protocol, for as long as the datagram will be active in the Internet.

Supposing the identifier value for a packet is *I*, there are *N capture nodes,* and the position of the current *capture node* in the *capture node* list is P_i, the packet is captured if and only if: $hash(I) \bmod N = P_i$

The main advantage of this approach is that synchronisation between the capturers is not as tight as in the round-robin approach. In this case, the *capture nodes* only need to have the topology synchronised, while the order of packets they see on the link does not have to be identical. As *topology update* messages arrive on a separate data channel, it might be possible that these messages are not received at the same time by all capture nodes, which may lead to an inconsistent view on the topology of *capture nodes* network. In order to address this issue each topology update message carries a *validity start* AVP which specifies from when the new topology should be used. This mechanism is provided in order to allow all capture nodes to switch the topology at the same time.

A disadvantage, however, may be the use of the identifier field, which may not lead to an equal distribution of capture responsibility among the different *capture nodes*. During the evaluation of DiCAP, tests have been performed to see how good the use of the identified field would be for balancing the load. The results of those tests are detailed in Evaluation (Section 5.2.3).

5.2.1.4 Operation Modes

The design of DiCAP allows for two modes of operation: a *distracted capture mode* and a *distribution mode*. The *distributed capture mode* follows the architecture described in Figure 5-1 while the *distribution mode* is depicted in Figure 5-5. In *distributed capture mode,* the *capture nodes* capture packet headers from the monitored link and send them to a *packet data analyser*. A drawback of this approach is that only the information in these packet headers is collected, while the payload is dropped. In order to address this issue, the *distribution mode* is introduced. In this operation mode the packet capture is not done in the DiCAP module, but by using other tools such as *libpcap*. DiCAP is only used to distribute the workload between several *libpcap* instances. In this mode, if a *capture node* decides to capture the current packet, its DiCAP module allows that packet to be forwarded to *libpcap*. Otherwise, the packet is dropped.

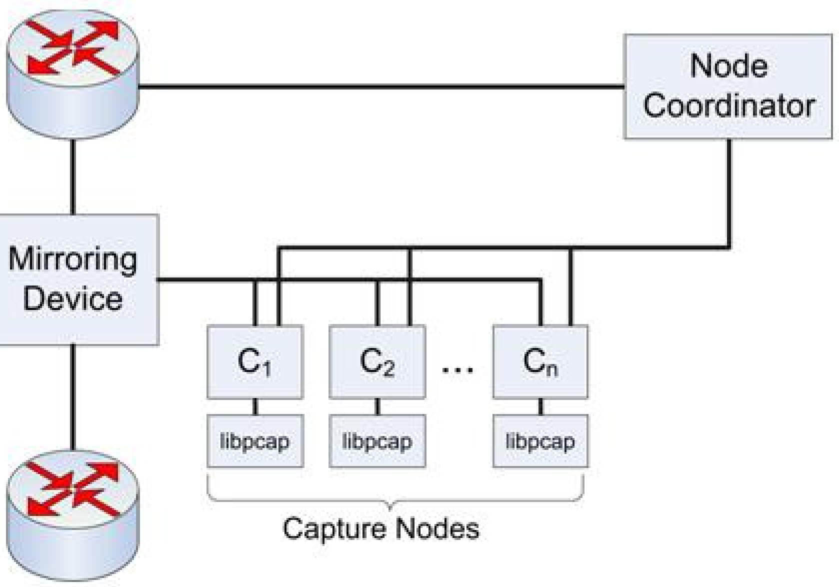


Figure 5-5. DiCAP in Distribution Mode

5.2.2 Implementation

The DiCAP prototype was implemented in C and C++ under Linux. It consists of a Linux kernel module, called DiCAP module, which runs on each capturing node, a coordinator node that is used to synchronise the capturing nodes, and a packet data analyser, which collects all the packet headers captured by the capture nodes. For implementing the Linux kernel module, version 2.6.16 of the Linux kernel was chosen.

In order to maximise the performance, the DiCAP kernel module was implemented as low as possible in the kernel networking stack, in order to drop as early as possible uninteresting packets, for reducing as much as possible packet the handling performed by the kernel. The architecture of a DiCAP-enabled Linux kernel is depicted in Figure 5-6.

Traditionally, each packet captured by the network interface card (NIC) is forwarded to the NIC driver, which allocates a memory structure (*sk_buff*) for the packet, copies the packet data in this memory structure, and then forwards the *sk_buff* data structure to the Linux kernel. The DiCAP module was placed within the NIC driver, so that packets for which the capturing node is not responsible are dropped before their corresponding *sk_buff* structure is created. Similar architectural approaches have been proposed for other network monitoring projects, such as [79], in order to achieve fast processing of network packets.

Having the DiCAP module called by the NIC driver makes the implementation device-dependent, but the modification of the driver is so small it can be easily ported to any other NIC driver. For the evaluation of DiCAP, the network card used was a 1 Gigabit Ethernet Broadcom Corporation NetXtreme BCM5721 card [80].

As Figure 5-6 shows, the DiCAP module consists of three main components: a *packet processor*, a *packet data forwarder*, and a *management unit*. As Operation Modes has shown, DiCAP may be re used in two modes: a *distribution capture mode* and a *distribution mode.* The implementation details for two modes are further detailed in the following subsections.

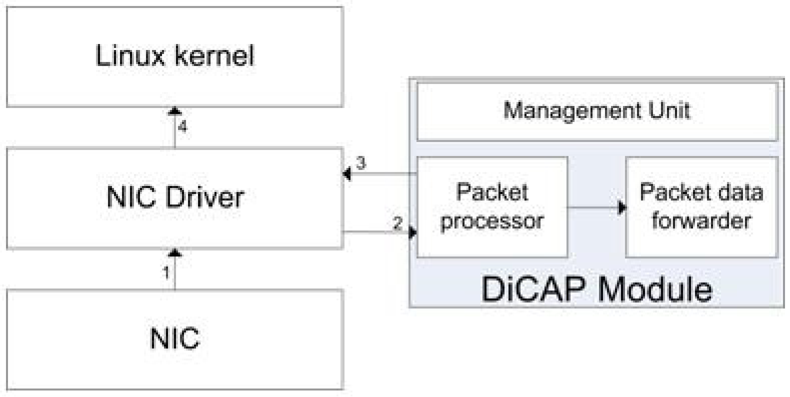


Figure 5-6. DiCAP Kernel Module.

5.2.2.1 Distributed Capture Mode

In the *distributed capture mode,* the DiCAP kernel module captures IP packet headers in a distributed manner and sends them to a packet data analyser.

The *packet processor* has the task of deciding whether a packet is going to be captured or not. It receives a packet from the NIC and first it checks on which interface the packet was received. If it was not received on the monitoring interface, the *packet processor* does not handle it and informs the NIC driver to deliver the packet to the Linux kernel. If the packet was received on the monitoring interface based on one of the rules described in Synchronisation (Section 5.2.1.3), each capture node will do one of the following:

- If the packet is to be captured locally, the packet data (in the prototype version the packet header) is sent to the *packet data forwarder,* while the NIC driver is instructed to drop the packet.
- If the packet is not within the responsibility of the current *capture node*, the NIC driver is instructed to drop the packet.

The *packet data forwarder* has the task of sending the captured data to a *packet analyser*. In the implemented prototype it stores in a buffer the first 52 bytes of each packet, including the IP and transport layer headers. Once the buffer is filled, it is sent via UDP to a *packet data analyser*. The reason for using UDP is to reduce as much as possible the burden on the capture nodes, so that most of the resources may be spent on capturing the packet headers. If TCP is desired to be used, in order to make sure that no packet headers are lost between the *capture nodes* and the *packet data analyser* node, the prototype can be easily extended to support TCP or SCTP between the *capture nodes* and the *packet data analyser*. In the current implementation, in each packet sent to the *packet data analyser* there are 28 packet headers.

The *management unit* performs maintenance tasks. It is implemented as a separate kernel thread. The *management unit* maintains communication with the node coordinator and gets from time to time the current topology of the capturing network, as well as information about the address (or addresses) of packet analysers.

5.2.2.2 Distribution Mode

When working in *distribution mode,* the DiCAP kernel module has the task of distributing the packet capturing task between several nodes but does not perform the packet capture. Instead, packet capture takes place in higher level libraries, such as *libpcap*. As a result, the *packet forwarder* is not needed, and thus not used, in this mode.

While the *management unit* has a similar task as in the *distributed capture mode*, the *packet processor* has a slightly different behaviour. For each incoming packet, the *packet processor* decides whether that packet is going to be locally processed or not. If the packet is going to be locally processed, the NIC driver is informed to forward the packet to the Linux kernel. Otherwise the NIC driver is informed to drop the packet.

5.2.3 Evaluation

In order to evaluate DiCAP, a set of performance tests have been performed. As the evaluation shows, DiCAP offers a considerable performance improvement for Linux-based packet capture applications. The goal of the evaluation was to investigate what are the limitations of existing capturing tools (*libpcap* and *libpcap-PFRING*) and how DiCAP compares to them. Additionally, for applications that require more than the packet headers, DiCAP was used in *distribution mode*, as a distributed capture coordinator for *libpcap*, and the combined performance of DiCAP and *libpcap* has been evaluated. The hardware used for evaluation was made of single-core Pentium 4 nodes each running at 3.6 GHz, each having 2 GB RAM and each being connected to two Gigabit Ethernet networks: one for receiving mirrored traffic and the other used for the communication with the coordinator node. The packet data analyser and the capture nodes shared a dedicated LAN, so the extra traffic did not affect the existing traffic on the mirrored link. The network traffic was generated using the Linux kernel packet generator *pktgen[81]*. In order to achieve the desired packet rates, two nodes have been used to generate traffic. In addition, the packet rate was also controlled by modifying the packet sizes.

The first experiment compares the capturing performance of *libpcap*, *libpcap-PF_RING* and *DiCAP* on a single node at different packet rates. For the tests involving th *libpcap* and *PF_RING* libraries, a sample application was developed, that counted the number of packets that were captured by those libraries. After each packet was captured and the counter incremented, the data was discarded. For testing *DiCAP*, the packet headers of each observed packet were sent to a *packet data analyser* node, which counted the number of packet headers received. As the purpose of the test was to evaluate the performance on a single node, DiCAP was configured to capture every packet observed on the link by a single node. Table 5-1 shows that the performance of both *libpcap* and *libpcap-PF_RING* is significantly affected by high packet rates. The loss rate in both cases was beyond 90% at 620.000 packets per second (pps). That rate was achieved by using 72 Byte packets, which filled about 38% of the link capacity (376 Mbps). *DiCAP* experienced no loss at any packet rate, showing it is the best solution out of the three evaluated for capturing packet headers at very high packet rates.

Table 5-1. Packet Loss during Capture on a Pentium 4 @ 3.6 GHz, 2 GB RAM, 1 GB Ethernet.

Packet Rate	libpcap Loss	PFRING Loss	DiCAP Loss
119 Kpps	0%	0%	0%
232 Kpps	10%	1%	0%
380 Kpps	75%	28%	0%
492 Kpps	90%	83%	0%
620 Kpps	93%	96%	0%

A second test was performed to check the empirical distribution of values for the identifier field of the IP packet header. As distribution of load in the capturing network is based on those values of the identifier field value, it is important to check if those values follow a uniform discrete distribution. For this test, traffic from a real network link was captured for two hours and a histogram was drawn. The total number of packets captured in that interval was 30 million packets. The result is shown in Figure 5-7. The result is slightly different than a roughly uniform distribution that was expected. The range of values for the identification field is 0 to 65355. The result of these test shows that two intervals cause the graphic not to show a uniform distribution function: 0 to 100 and 47.000 to 54.000.

One conclusion that can be drawn from these results is that using the IP identification field in order to decide which packets to capture at each capture node may not necessarily lead to a perfect load balance. However, the approach is still acceptable, as the total number of packets that were outside the uniform distribution function was below 5%. This observation does not impact other performance results of DiCAP, since the DiCAP prototype used for evaluation was based on the round-robin selection approach.

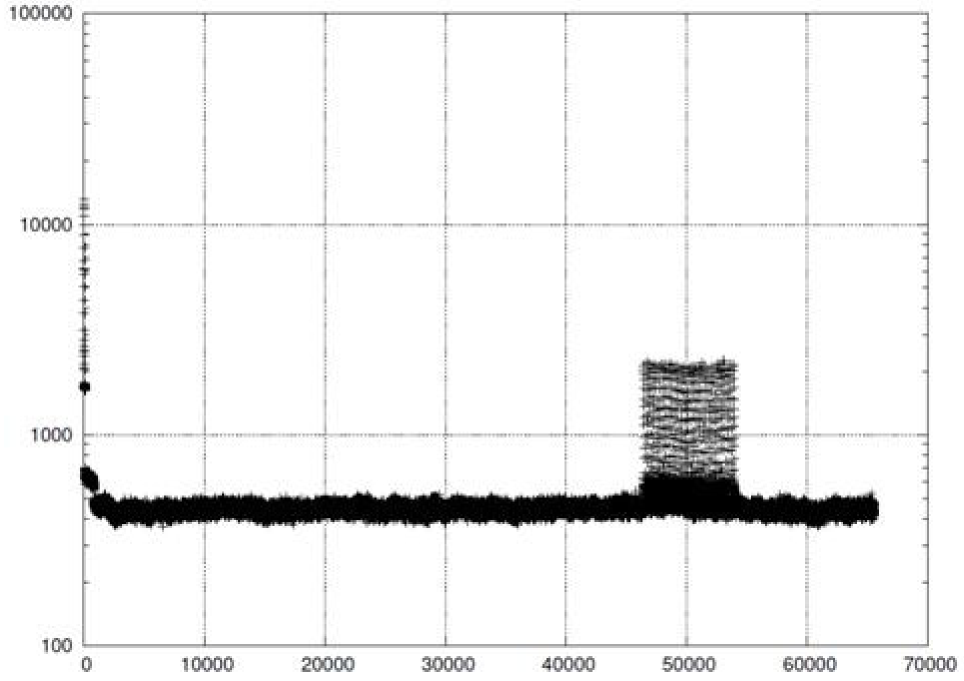


Figure 5-7. IP Identifier Field Value Histogram

The third test investigates how *DiCAP* running in *distribution mode* works in combination with *libpcap.* During this test, the capturing capability of DiCAP has been disabled. Instead of capturing the packet header, in this test DiCAP lets the selected packets pass through the kernel, so that libpcap can capture them. Three tests have been performed, each using different packet sizes: 40 Byte, 256 Byte, and 512 Byte. The corresponding packet rates were: 625 Kpps, 480 Kpps, and 270 Kpps.

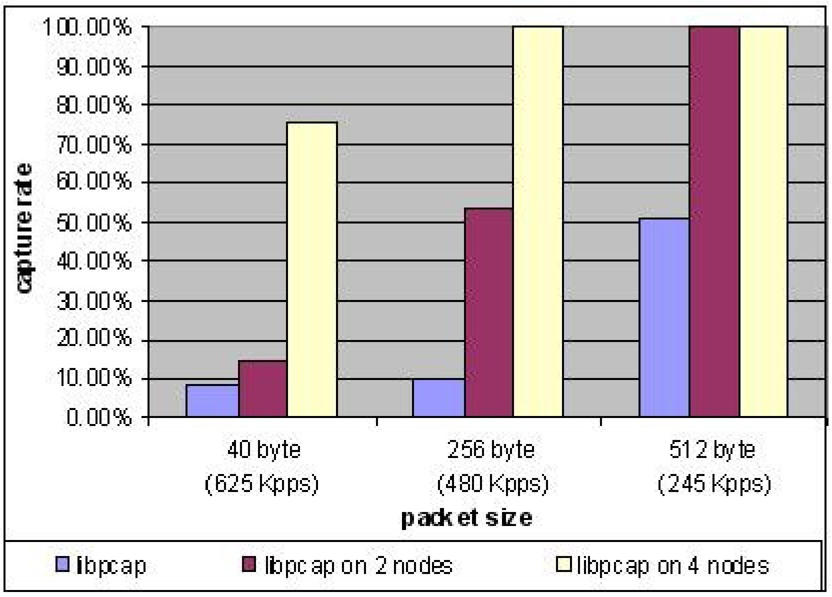


Figure 5-8. Performance Improvement of libpcap.

Each test consisted out of three parts. In the first measurement it was observed what percentage of the total number of packets sent by the packet generator are received by the *libpcap* application running on one capturing node with an unmodified Linux kernel. In the second measurement, a setup with two capturing nodes, each running DiCAP and libpcap, was prepared and the percentage of the captured packets was recorded. The third measurement was performed similarly to the second one, but with four capture nodes instead of two. The results of this test are shown in Figure 5-8.

Figure 5-8 shows that with a traditional libpcap the maximum number of packets captured is around 50% for packet sizes of 512 Byte. Lower loss rates were expected at larger packet sizes as the packet rate for these is lower. It can be observed that with two capture nodes running libpcap in parallel using DiCAP the capture rate was doubled for small packets and was increased more than 5 times for packet sizes of 256 Byte.

With just two capture nodes it was possible to capture all packets of 512 Byte. With four capture nodes running in parallel libpcap the capture rate for very small packets (40 Byte) increased tenfold while for 256 Byte packets the capture rate was 100%. Another test not showed in the figure shows that with five parallel capture nodes a capture rate of 100% can be achieved even for the 40 Byte packet sizes.

As the first test shows DiCAP can perform loss-less packet capture at high-speed packet rates using Linux. If one may say that the usability of the system in such a way is reduced as just the IP and transport header values are captured, the third test shows that DiCAP may be used easily with other capture tools in order to highly improve their performance. The authors see the main advantage of DiCAP not only in the observed performance boosts, but also in its scalable architecture that allows for combined resources of multiple nodes to be used together more efficiently.

5.2.4 Conclusions

This section presented the design, prototypical implementation, and evaluation of *DiCAP*, an architecture for distributed IP packet capturing. *DiCAP* does not require any dedicated hardware, which makes it a cost-effective solution for capturing IP packet headers at high packet rates. The evaluation results show that when used for capturing IP packet headers on a single machine, DiCAP experiences no loss at high packet rates, whereas *libpcap* and *libpcap-RING* experience up to 96% packet loss at the same packet rates.

An important characteristic of *DiCAP* is the possibility of being used in parallel with other packet capturing tools, such as *libpcap* or *libpcap_PFRING*, in order to increase their performance, by distributing their workload across several nodes. Such a combination is required for scenarios in which payload data is required to be analysed. In such a setup, the evaluation has shown performance increases of up to 500%, when two capture nodes, using *libpcap* and DiCAP, were used in parallel, and up to 700%, when four *capture nodes* were used in parallel. Being implemented as a LINUX open-source project, DiCAP can easily be extended with further functionality.

The scalable architecture of DiCAP allows network professionals dealing with network measurements to increase strongly the performance of their measurements by adding new resources into the packet capturing infrastructure. The simplicity of the design allows DiCAP to be easily implemented in hardware, leading towards a hardware dedicated packet capture cluster architecture.

Further improvement of DiCAP performance might be achieved by making use of the existing ring buffers on the network interface cards. Future work shall investigate how the integration of those ring buffers into the DiCAP architecture may improve its performance. In addition, future work shall also concentrate on finding more suitable hashing functions for selection of packets.

5.3 DIPStorage: Distributed Storage of IP Flow Records

**A copy of this report has been published in [82].*

The measurement of IP traffic generated by a device connected to an IP network is the first step of several key operations in network management: load balancing, intrusion detection, performance monitoring, generation of traffic statistics, or charging for connection time, bandwidth used or volume transferred. Depending on the granularity of the data collected as well as on the type of link on which traffic is measured the information gathered may range from a few kilobytes per hour up to gigabytes per second. One of the main mechanisms to retrieve information about network traffic is flow accounting. Flow accounting is concerned with measuring the IP traffic on a per flow basis. A flow is typically identified as a unidirectional sequence of packets from one source point to one destination point having one or several common IP header fields. The most commonly used fields to identify an IP flow are: source IP address, destination IP address, IP protocol number, source port number, and destination port number. The IP header fields that define an IP flow are also called flow keys. In IP flow accounting during the traffic measurement process, at least the following information is collected for each IP flow: number of packets in the flow, total bytes transferred, start time of the flow, end time of the flow, TCP flags observed, incoming and outgoing router interface.

Today, the storage of IP flow records is mostly done by centralised components. Since traffic volumes increase, such centralised solutions can no longer cope completely with the resulting data increase, so they use sampling as a mechanism to reduce data volumes. Therefore, an alternative strategy for processing such high data volumes in future is needed. Using Peer-to-Peer (P2P) mechanisms for the development of storage platforms has shown that reliability and robustness of centralised platforms can be highly improved [83]. Storage of IP flow records is very challenging, mainly because a retrieval operation is typically preceded by a complex search operation, thus, distributing the storage (and search operations as well) should greatly improve the performance of a query operation.

The IETF is currently standardising the IPFIX [76] protocol as well as requirements for IP flow storage. These specifications define different templates to be used for flow records. DIPStorage is not constrained by a particular IP flow export protocol. The prototype implemented, however, uses NetFlow version 5 [84] as the export protocol for IP flow records. A change to the IPFIX protocol is foreseen for the near future and will require minimal change in the existing prototype.

The main goal of this work was to investigate, whether P2P mechanisms that proved efficient in other application areas can be used to improve storage scalability and query performance of IP flow storage by distributing these tasks to multiple nodes. Distribution in the context of this work does not imply (but does not exclude either) geographic distribution. Nodes may be collocated within the same data centre or distributed within the network of an ISP. DIPStorage provides the underlying mechanisms that allow multiple nodes to share their resources for storing IP flow records. In order to achieve this goal, a prototype was designed, implemented, and evaluated. The evaluation shows that DIPStorage meets the desired goal. Due to the fact that the distributed storage platform was based on P2P mechanisms, traditional database (centralised or distributed) systems have not been evaluated.

5.3.1 DIPStorage Architecture

Existing centralised solutions to IP flow collection have several major drawbacks: they lack storage scalability, suffer from a single point of failure, and flow retrieving performance degrades as the number of stored IP flow records increases. P2P systems have proven to be an efficient solution to scalability and reliability [83] problems of centralised storage systems. Since all nodes are “equal”, the tasks of a node, which leaves the network, may be taken over by any other node. In order to benefit from advantages of P2P applications, the architecture of DIPStorage is built on P2P concepts. In general, DIPStorage does not require that nodes are geographically distributed, but it provides a platform for sharing resources for storing IP flow records.

The main component of DIPStorage architecture is a Distributed Hash Table (DHT). This DHT is the enabler of DIPStorage. The way it is organised highly influences the way flow records are distributed in the storage network. Each node of the DHT is responsible for storing of a subset of flow records. For each IP flow record received by the storage platform a 64 bit flow ID is calculated by applying a hash function on the IP flow keys. Each node is responsible for the storage of a range of flow IDs. Based on its flow ID each IP flow record is routed through the storage platform to the node responsible for its storage.

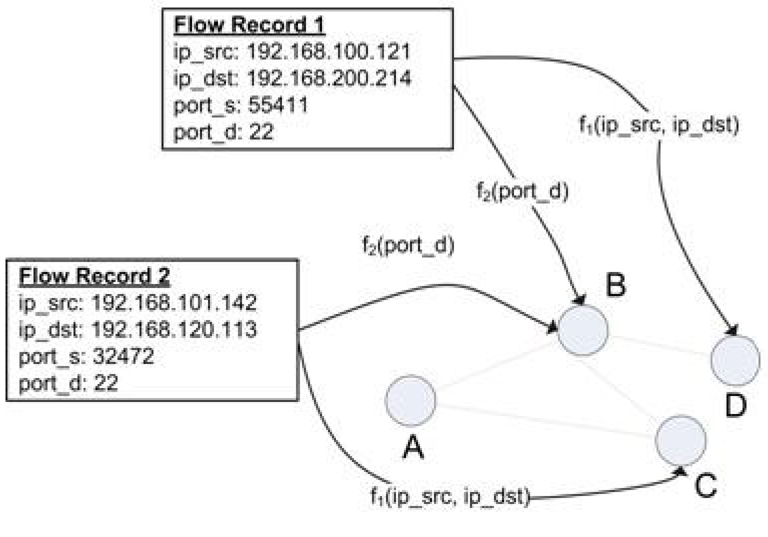


Figure 5-9. Flow Storage in Structured Storage Strategy.

Two strategies have been identified for routing an IP flow record within the storage platform: random storage strategy and structured storage strategy. In the random storage strategy the flow ID for each IP flow is randomly generated. By the random strategy approach flow IDs are uniformly distributed in the [0-2^{64}-1] interval. If this interval is split in equal parts and each of the storage nodes is responsible for one of those intervals, the storage load is distributed evenly between storage nodes. However, the main disadvantage of this approach is that each query for IP flow records needs to be broadcasted to all nodes and processed by all storage nodes. Therefore, the structured storage approach addresses this drawback by grouping IP flow records with similar flow keys close with respect to their flow ID. Figure 5-9 shows the effect of how structured storage strategies store two IP flow records with similar characteristics. The figure shows two flow records representing two distinct secure shell (*ssh*) flows having a common destination port number (22), but different source and destination IP addresses. In the example above, two hash functions (f_1 and f_2) are used for storing each flow record twice. By using multiple hash functions for storing a single flow record, redundancy is introduced in order to achieve higher fault tolerance. As f_1 is a function of IP source address and IP destination address, hash results of this function for the two flow records in the above example are different. This is shown in the figure by the storage of the two flow records on different nodes (C and D). Since f_2 generates a hash code based only on the destination port number, which is the same for the two IP flows in the example, the resulting hash codes for the two flow records are the same so both flow records will be stored on the same node. Therefore, DIPStorage is based on the structured storage strategy.

5.3.1.1 Tank-based View

For storing data DIPStorage uses attributes from IP flow records, which include IP addresses and ports from both source and destination. As observed in the example shown in Figure 5-9, the chosen hash function and the flow keys to which it is applied highly influence the way the IP flow records with similar flow keys are distributed in the storage network.

DIPStorage establishes the idea of a tank-based view (Figure 5-10). A *Tank* is a subset of peers which form a group that stores IP flow records under a specific set of rules. These rules include a hash function and the corresponding flow keys to which it is applied.

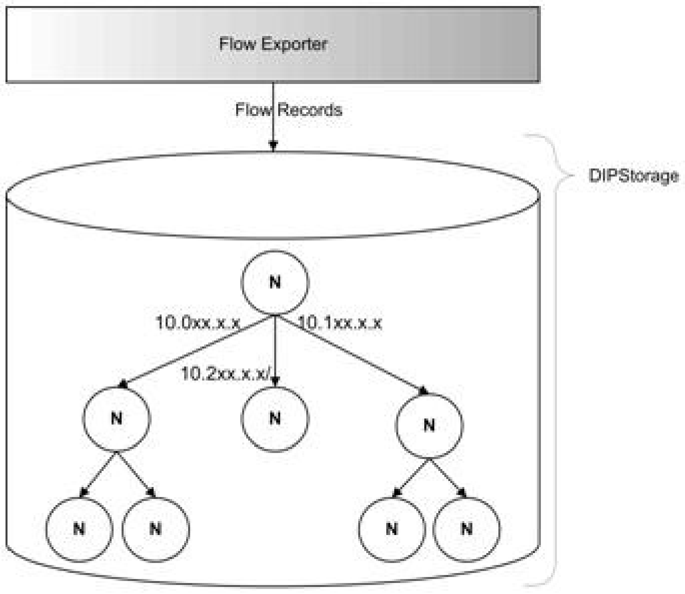


Figure 5-10. Tank-based View.

Drawing from the ideas of JXTA (Juxtapose) [85], a single tank forms a group of interest that actually stores incoming data based on a specific attribute. A single tank is able to completely handle incoming data and queries on it's own without requiring other tanks to be working. A single tank is being constructed using a tank indexer (TI) to manage the nodes in the respective tank. Although all nodes are equal within a tank group, they assume data storage responsibilities in a treelike fashion. Depending on available nodes the tank may have multiple levels, where each level refines the granularity of the data stored in the subtree. The more nodes or respectively peers are available, the more fine-grained the data routing can be. In case the storage tank uses the source IP address of the IP flow record as its storage attribute, routing is done by splitting the IP range by the number of nodes available on each level. In the IP case, the first level splits the first part of the IP address, the second the second part and so on. Whenever a node receives a flow record it calculates its flow ID and decides, whether it is responsible for that flow record. If not, then it calculates — based on the flow ID and the known children — the next hop where to send the flow record to. Since different tanks in DIPStorage have different storage rules, search queries can be efficiently routed through the storage network to find data according to an attribute (for example the IP source address). More specifically, nodes within a storage tank can be queried directly, because their parent tank indexer knows exactly where data may be stored and therefore redirects the query to a specific subtree. In case of multiple levels involved, every parent of a subtree sends the query down to its matching children. It waits for all answers, aggregates them, and sends them to its own parent. In order to avoid deadlocks, a time-out prevents a node to wait indefinitely long for an answer from one of its children. If the time-out is reached, the parent node can initiate any administrative tasks needed to rejoin the missing node or to recover the data lost. By introducing levels of responsibilities, the system is able to handle failovers efficiently. If a parent node fails, the second level of responsibility is delegated toward child nodes, which shall elect a replacement for the missing node. All child nodes during the election process need to answer incoming queries all together.

5.3.1.2 Multi-Tank Organisation

Since a single data tank stores IP flow records based on a particular IP flow attribute each query that does not contain that attribute cannot be optimised. This happens because the query needs to be forwarded to all nodes in all sub-trees as the information required to route the query is missing. In order to address this issue, DIPStorage uses several data tanks for storing the IP flow records under different rules. Figure 5-11 shows how four data tanks are used by DIPStorage to store the IP flow records based on: source IP address, destination IP address, source port number, destination port number. Based on the query received, the query analyser decides which data tank is best optimised for answering the query, and then forwards the query to that tank.

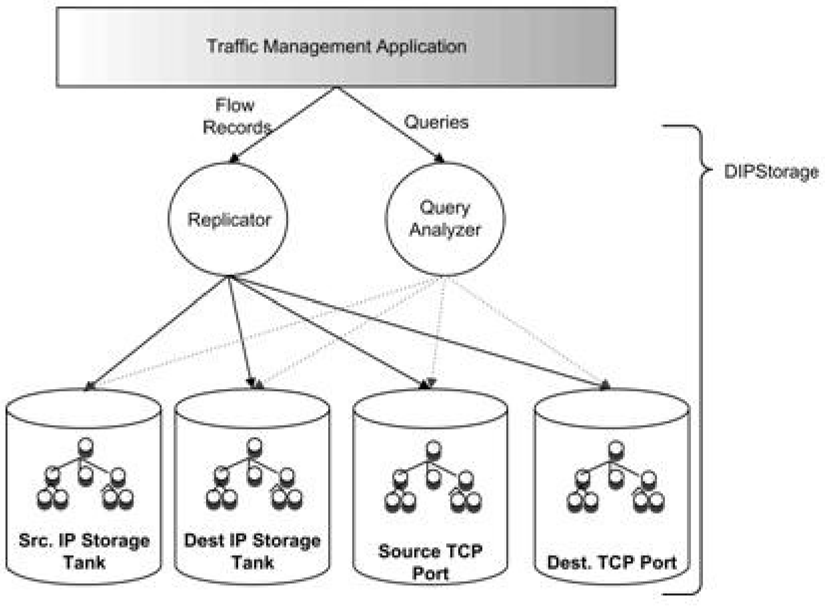


Figure 5-11. Multi-Tank View.

A traffic management application deals with the generation of flow records and their delivery to DIPStorage as well as with creation of queries of IP flow records. For example, a router with NetFlow export capabilities could be used to generate IP flow records. The flow replicator is responsible for distributing incoming data evenly across the tanks. Every tank stores each IP flow record according to its internal distribution rules. Subsequent queries for data can be handled by the appropriate data tank. In case one of the tanks fails to provide the requested data, other tanks can still potentially answer the query, although in that case the overall query time may be higher. The query analyser has the task to decide which data tank a query is routed to such that the query is answered efficiently. Multiple tank strategies also provide redundancy. In case any node or a whole tank fails, other tanks have the ability to get the lost data and send it for duplication in the affected tank.

5.3.1.3 Load Balancing

One of the reasons for distributing IP flow records to different nodes is to decrease the query time in case of large IP flow records databases. Preferably the data should be equally distributed among participating nodes, so that response times for any two queries are similar. As the flow storage responsibility is based on a hash value of the IP flow record, the load of a node can be increased or decreased by changing the range of hash values for which that node is responsible. A load balancing process is started whenever a node is detected to store considerably more flow records then an overall average. As nodes may be added or removed arbitrarily from DIPStorage, a load balance mechanism also allows for redistribution of work in these situations.

5.3.1.4 Redundancy

One key concept of DIPStorage is the storage of each IP flow record in several data tanks. Such an approach has a twofold advantage: on one hand it achieves a better fault tolerance by redundant storage of each IP flow record, while on the other hand it improves the query efficiency by optimising each data tank for a particular type of queries.

Since each flow record is stored more than once, if a storage node disappears from the storage network, its data is not lost from the system, but only from its data tank. Once a node is detected as “down“, its data tank requests one of the other data tanks the missing data. For doing that, the data tank that detects a missing node first calculates the range of IP flow records for which the missing node was responsible. Afterwards the data tank performs a rebalancing of the load, so that the missing range is reassigned to other nodes in the data tank. Once the load balancing is done, the data tank asks one of the other tanks for all flow records within the missing range. Using this technique, an IP flow record is lost only if the respective responsible nodes in every tank are lost in a short time interval.

The second advantage of having redundant data, which is organised under different rules, is to optimise different types of queries. For example, a query such as “get all IP flow records which have the source IP address 10.100.100.1“ can be answered very fast by the data tank 1 (Figure 5-11), because the query can be routed directly to the node that stores flow records with that source IP addresses. However, a query such as ”get all flow records with the destination port 22“ asked to data tank 1 would have the effect of being broadcast to all nodes in the data tank. This second query can be answered more efficiently by data tank 4, since it stores IP flow records based on destination port numbers. Such behaviour would be implemented in a traditional databases approach by using index files. In the presented scenario, however, such index files would be too expensive to keep due to the time required for the update operation.

5.3.2 Implementation

The implementation of DIPStorage follows a layered approach as shown in Figure 5-12. The P2P layer of DIPStorage provides the ability to handle a P2P network on top of Pastry technology, and further more, establishes a second overlay for building a hybrid super peer network. The P2P layer is implemented on top of Freepastry [86], which is a JAVA-based framework, for building Pastry networks. On top of Pastry DIPStorage uses an add-on called Scribe [87], which is also a part of the Freepastry framework.

The P2P layer also contains all messages that need to be sent from one node to another using either Pastry directly or the multicast mechanisms of Scribe. Therefore, a message either implements the *Message Class* from Pastry, or the *ScribeContent Class* from Scribe. If a message travels through Scribe and Pastry, it primarily implements *ScribeContent* and is then packed into a Pastry Message when sent directly through Pastry. Most messages either form a request for an arbitrary task or a response to it. The routing layer handles two tasks. It routes incoming data to the appropriate storage client and queries the system efficiently. The routing of IP flow records is based on IP flow attributes. Each node knows where to route an IP flow record or query based on flow keys.

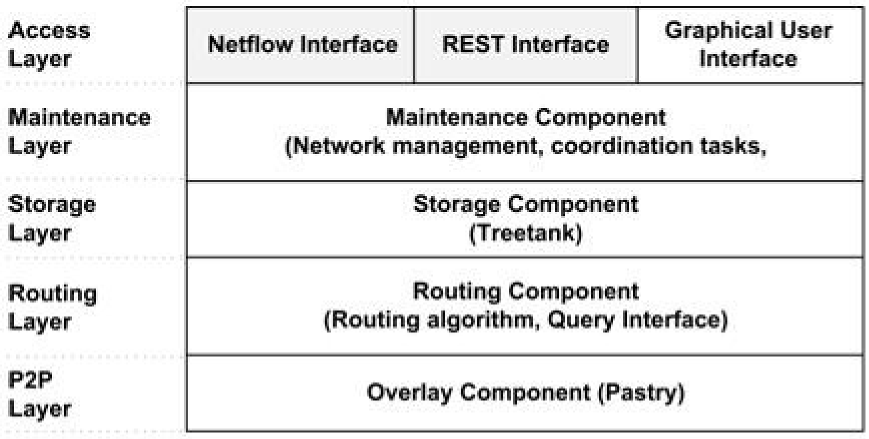

Access Layer
Netflow Interface
REST Interface
Graphical User Interface
Maintenance Layer
Maintenance Component (Network management, coordination tasks,
Storage Layer
Storage Component (Treetank)
Routing Layer
Routing Component (Routing algorithm, Query Interface)
P2P Layer
Overlay Component (Pastry)

Figure 5-12. DIPStorage Layers.

The storage layer includes the main storage component of DIPStorage. The main intention was to use an existing system that is optimised for storage and query efficiency. One of the bigger open-source database systems available, MySQL [88], introduces drawbacks. First, it requires a node to have a running MySQL installation. Second, it does not enable easily replication. Furthermore, the scheme of the database has to be designed in advance and embedding it requires the usage of SQL language (neglecting the existence of persistence frameworks). This is why DIPStorage uses a JAVA-based XML Database called TreeTank [89], which enables DIPStorage to provide easy replication and standardised querying through XPath [90] queries. The XML nature of that database enables DIPStorage to either store the data in a treelike structure on physical nodes, when enough nodes are available, or to build a XML based tree structure on one node. Additionally, traditional NetFlow collector engines, such as nfDump or flow-tools, may be used as storage components. In such a case DIPStorage would serve as an enabler for a distributed operation of those tools.

Basic tasks of the maintenance layer are the self organisation of the underlying P2P network. The first node to join DIPStorage starts acting as a Bootstrap Server. Nodes within DIPStorage do not need necessarily to know the IP address of a Bootstrap Server, but can get relevant service information through the MulticastDNS [91], which is implemented in DIPStorage using JmDNS [92]. Starting with the second node, each participant generates an identifier and connects to a Bootstrap Server. There may be multiple nodes acting as a Bootstrap Server in order to load-balance the workload or for resilience against failovers. One of the duties of a Bootstrap Server is the assignment of tasks to new nodes. As soon as a sufficient number of nodes are ready to form one main tank, the Tank Coordinator will enable DIPStorage to receive and query data by adding middleware nodes.

5.3.3 Evaluation

In order to evaluate DIPStorage, a functional evaluation as well as a performance evaluation was performed. For doing the tests, a set of 6 to 15 nodes have been used in different DIPStorage configurations. During all the tests nodes were connected via a single Local Area Network. The functional evaluation shall show that the prototype can build a storage network, can perform P2P node management within the storage network, and can route IP flow records and queries within the data tanks. Using P2P mechanisms the DIPStorage prototype may be used to build a scalable storage platform for IP flow records. The current version of the prototype, however, does not replicate IP flow records, and only uses a single data tank for storage. The addition of this functionality is foreseen for a next version of the prototype. The second part of the evaluation investigated how much query performance improvement DIPStorage achieves in comparison to a centralised solution. During the performance evaluation two different distributed storage strategies (random approach and structured approach) have been evaluated in order to see which one performs better under load. For these tests the storage network was not idle, but was receiving flow records at a constant rate.

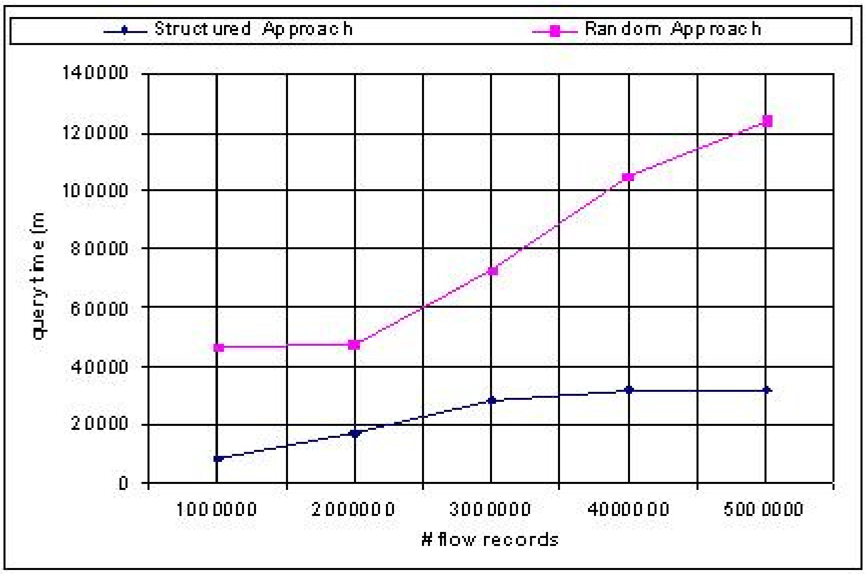


Figure 5-13. Distributed Storage Strategies Comparison.

The test consisted out of a set of queries running in parallel. The results presented in Figure 5-13 show that the structured storage strategy outperforms the random storage.

The main reason is that in the structured approach the query is routed directly to the responsible storage node, which makes a lookup in its local database, while in the random storage the query is received by all nodes and all of them have to perform a local lookup. In case of multiple different queries running in parallel, each query is processed by a different node when using the structured approach. When random strategy is used, all nodes need to process all queries.

The second performance test compares the performance of the structured distributed storage with the performance of a centralised storage system. The centralised storage system consists out of a single storage component that stores all flow records received locally. The storage component for the centralised storage system was identical to the one used by the nodes in the distributed approach. This comparison measured the performance increase of work distribution. Results are shown in Figure 5-14. For 357 million records, the increasing ratio between query time in the centralised approach and the distributed approach shown in the second test demonstrates that with more flow records added to the storage network the single node performance will deteriorate faster compared to the distributed structured approach. This is because a single node needs to query all the IP flow records, while in the distributed approach each node needs to query a smaller number of flow records. The ratio between the single node and the distributed structured version increases as more nodes are added to the storage network as in the latter case each node has to store less data.

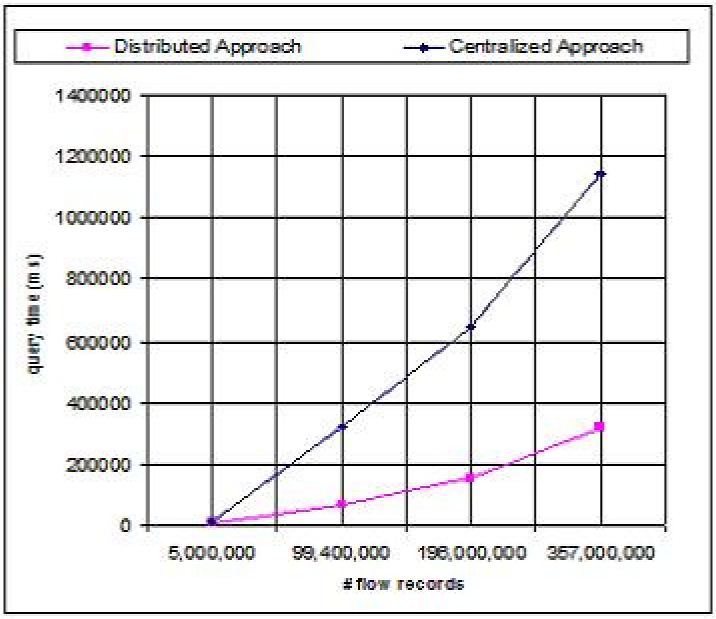


Figure 5-14. Centralised Approach vs. Distributed Approach.

It is interesting to see, how the DIPStorage approach handles load. A centralised component may handle loads not very well, because storage and query tasks, as well as administrative tasks need to be handled by one single component. In the distributed approach, these tasks are split onto multiple peers. The evaluation of DIPStorage includes a differentiation on the performance between a storage network under load and an idle storage network with respect to query performance.

As Figure 5-15 shows, the difference in performance between these two cases considered is minimal. This means that the distributed approach used in DIPStorage can handle a high load in a fairly efficient way, which gives confidence in the system to grow with future needs.

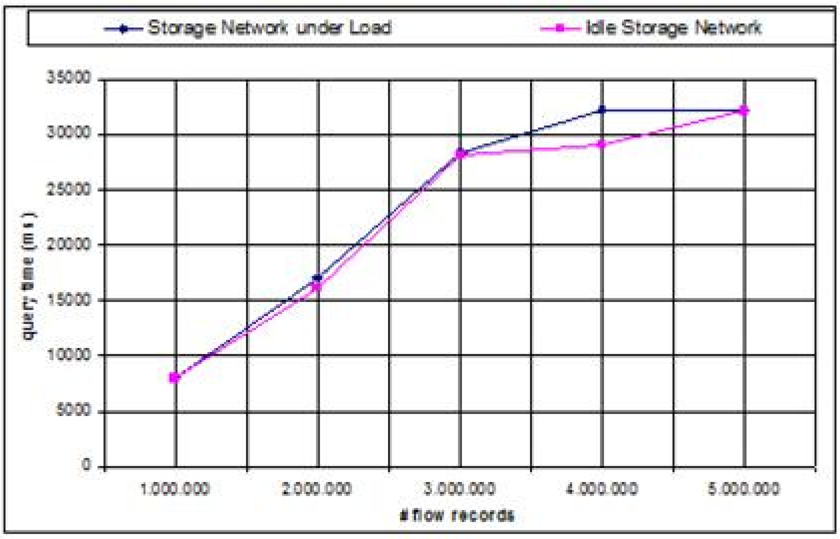


Figure 5-15. Load Resilience in Distributed Approach.

5.3.4 Conclusions

With the continuing increase of available data rates on backbone network links, the amount of IP flow records network providers have to handle increases as well. Centralised IP flow records storage systems, although easy to maintain, since all the data is stored in a single place, do not scale easily with respect to the storage size they can achieve, as well as with respect to the query performance they need. Often providers need to sample traffic in order to reduce the amount of data they have to handle. A distributed storage platform for IP flow records — as the one presented here — allows for a provider to increase the amount of storage space by adding new storage nodes. Moreover, the performance of IP flow record processing can be improved also by using more computational resources during the retrieval of IP flow records from the storage space.

Therefore, the evaluation of the prototype implemented shows that a P2P-based distributed storage system for IP flow records is feasible. This prototypical implementation proves that storage distribution highly reduces the query time for IP flow records and can be applied in real systems.

Since most of the functionality on DIPStorage is in place, future work will be concentrated on improving the performance of the storage and query system as well as on the extension to support more complex applications, such as detection of route asymmetries based on IP flow records.

Finally, the conclusion is drawn that DIPStorage can help network operators to build a scalable storage and analysis platform for IP flow records with low costs. Moreover, being able to store more IP flow records and achieving better performance querying them, new application scenarios using IP flow records can be implemented on top of DIPStorage.

6 Federated Management in the Future Internet (FeMaFI)

6.1 Introduction

Networks are becoming service-aware. Service awareness means not only that all digital items pertaining to a service are delivered but also that all business or other relations pertaining to a service offer are fulfilled and the network resources are optimally used in the service delivery. Future Internet Networks (FINs) are envisaged in Autonomic Internet [93], FIND program [94], GENI program [95], Ambient Networks [96], Ad-hoc networks [97], AKARI [98] and others. These initiatives consider time varying topology networks implying that service availability and context change as nodes join and leave the network. In addition, service and network providers must have means to collaborate and to form collaborative execution environments so that network and service providers can form coalitions targeting the provision of composite services. All in all, complexity will grow with service plurality leading to new requirements on control and management of service provisioning.

In the Future Internet Network's (FINs) paradigm it becomes mandatory for network and service providers to offer and publish their services so that more complex services could be provided. In such scenes, virtual service broker negotiators would take care of the service requests and should provide support to organise the underlying service providers as to provide the appropriate composition tasks. This support should be carried out taking into account the plethora of service providers, the services they will to provide, their interests, their service qualities and other key aspects. All in all, this functionality should be provided leveraging the requester entities from complex decision making processes. For this the above virtual service negotiation broker should implement a consistent and distributed service coalition mechanism. This mechanism should be scalable and must consider the new degree of automation and self-management of the envisaged design of the FINs.

Federated Management in the Future Internet (FeMaFI) project proposes a mechanism that provides support for the service coalition formation process for Future Internet virtual service negotiation brokers. We propose an algorithm that contemplates several aspects that we have envisaged necessary for FIN-oriented solutions such as that proposed within the Autonomic Internet approach [93].

In addition to a service requester, the algorithm considers the decomposition of a service into sub-services that should be provided by a number of underlying service providers. Each service provider is characterised with a set of capabilities, and different performance levels for each capability. Finally a coalition of providers noted as an array of members satisfying the task is the result of the algorithm execution. We describe and analyse the technical aspects of the algorithm including its interfaces and functionality within the Autonomic Internet approach [93], and other considerations that the later benefits from like the dynamic service composition, scalability, distribution features and transparency to service providers and end-users. The conclusions drawn from our simulation experiments is that the proposed algorithm is a feasible and suitable alternative to address the challenging requirements on management and control for the service composition paradigm of the FINs and in particular of the Autonomic Orchestration Overlay (AOO) developed in the AutoI framework.

This report is organised as follows. Section 6.2 provides some background information of the concepts addressed in this project. Section 6.3 describes the proposed algorithm. Section 6.4 provides a basic scenario and experimental validation results with larger scale scenarios are provided in Section 6.5. Finally, Section 6.6 concludes the report.

6.2 Background and Application Context

6.2.1 Social Networks

A social network is one where different entities intervene and that are capable to communicate and share information. Analysis of social networks [99] has been applied to a number of fields like sociology, anthropology, communication and organisational studies, economics, biology, etc., with the aim to study the different types of relations that can be established amongst societies, network topologies, dynamicity of networks and so forth. A few technical parameters of social network related to our study are: node – representation of an entity that intervene in the social network and that is capable to communicate and share information (see circled nodes in Figure 6-1); edge – bi-directional connection sight between service providers (see edge examples in Figure 6-1); average degree – mean number of neighbours in each node (see degree example in Figure 6-2; the maximum degree is 3, the minimum degree is 0, and the mean degree is 2); path length – distance between pairs of nodes in the network (see Figure 6-2); average path length – average of distances between all nodes in the network.

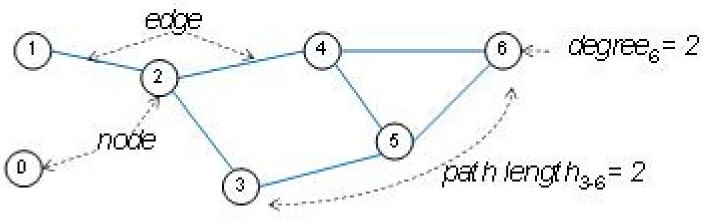


Figure 6-1. Social Network basics.

With the exponential growth of communication networks the social networks paradigm is not only studied in the societal aspects but also in the information area. Web search research is an area that has taken advantage of the social network theory, exploiting the similarities between the web and social networks. More precise search engines and more effective data mining algorithms have been created, exploiting social network concepts and methodologies [100].

Another area that has taken advantage of the social network paradigm is that of Multi-agent Systems where agents need to interact in order to fulfil their common or individual goals. In some cases, an agent does not need to interact with every other agent, and social knowledge can be spread in the basis of some neighbourhood model such as functional neighbouring (as in supply chain models [101]), or geographical neighbouring (as in sensor networks [102]). In other cases, agents might benefit from being aware of every other agent of the population (farsighted social knowledge) as in organisational systems [103], [104], [105] where agents have to explore a search space of agent group combinations in order to improve data flow, allocate resources efficiently, solve a problem in a coordinated manner, or to improve their outcomes by creating alliances. A key issue here that surrounds our study is that a fallback position to the combinatorial explosion of social farsightedness is for nodes only to interact with near neighbours initially and attempt to use these connections to find other nodes if necessary.

6.2.2 Service Provider Coalitions

Virtual Enterprises (VEs) are temporary federations of autonomous, legally and economically independent Service Providers that collaborate on common business goals, taking place in electronic “virtual” service spaces [106]. A successful VE requires a very close co-operation among all Service Providers, as with a single company. The above reasons highlight VE’s need for highly dynamic and flexible information infrastructures, procedures and algorithms that enable such VE to provide services to its customers.

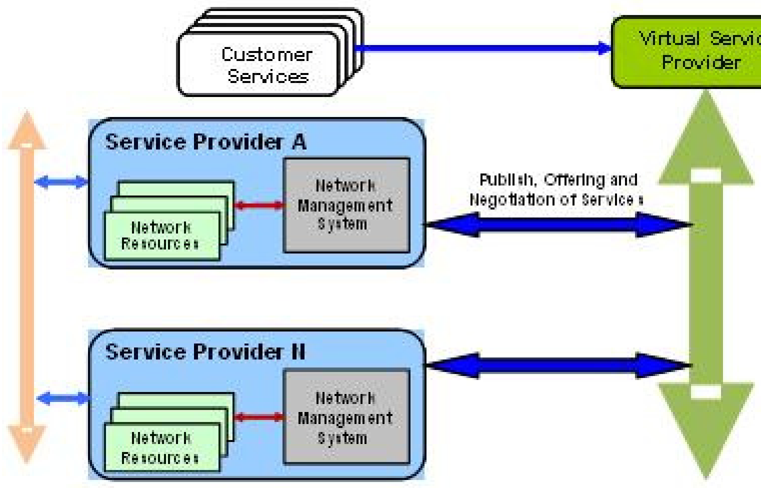


Figure 6-2. Framework for the service marketplace coalitions.

A number of collaborative Service Providers in this context form a Virtual Service Provider (VSP, see Figure 6-2). Relevant to our study are the key functionalities of a VSP as support for the search for suitable Service Providers, negotiation among these Service Providers, and configuration of the selected Service Providers according to business objectives and the customer services demands.

In order for the negotiation process to take place, each Service Provider must have a management system that allows such service negotiations to happen. In turn, the Virtual Service Provider must provide support for a service marketplace [106] in which each Service Provider publishes and offers their services. The service marketplace should provide support for service coalitional actions. The above highlights the need for an optimal negotiation mechanism that provides support for the service coalition formation process.

6.2.3 AutoI Autonomic Orchestration Contextual Framework

The long-term aim of our work is to conceive a service market place that can support the provisioning of services in Future Internet Networks (FINs), and hence that can be part of a large service management system for this purpose. This section analyses how the algorithm addresses key virtual service provisioning requirements for this purpose.

The FIN-oriented approach in which we analyse the algorithm relies on the framework developed in the context of the EU IST AUTOI – Autonomic Internet project [93]. The relevant elements of the AutoI architectural model that are influenced by the algorithm proposed in this report are shown in Figure 6-3. In the following paragraphs we provide a summary description of this architecture. More specific details can be found in [93]. The Orchestration Plane provides assistance for the Service Lifecycle Management, namely during the actual creation, deployment, activation, modification and in general, any operation related to the application services or management services.

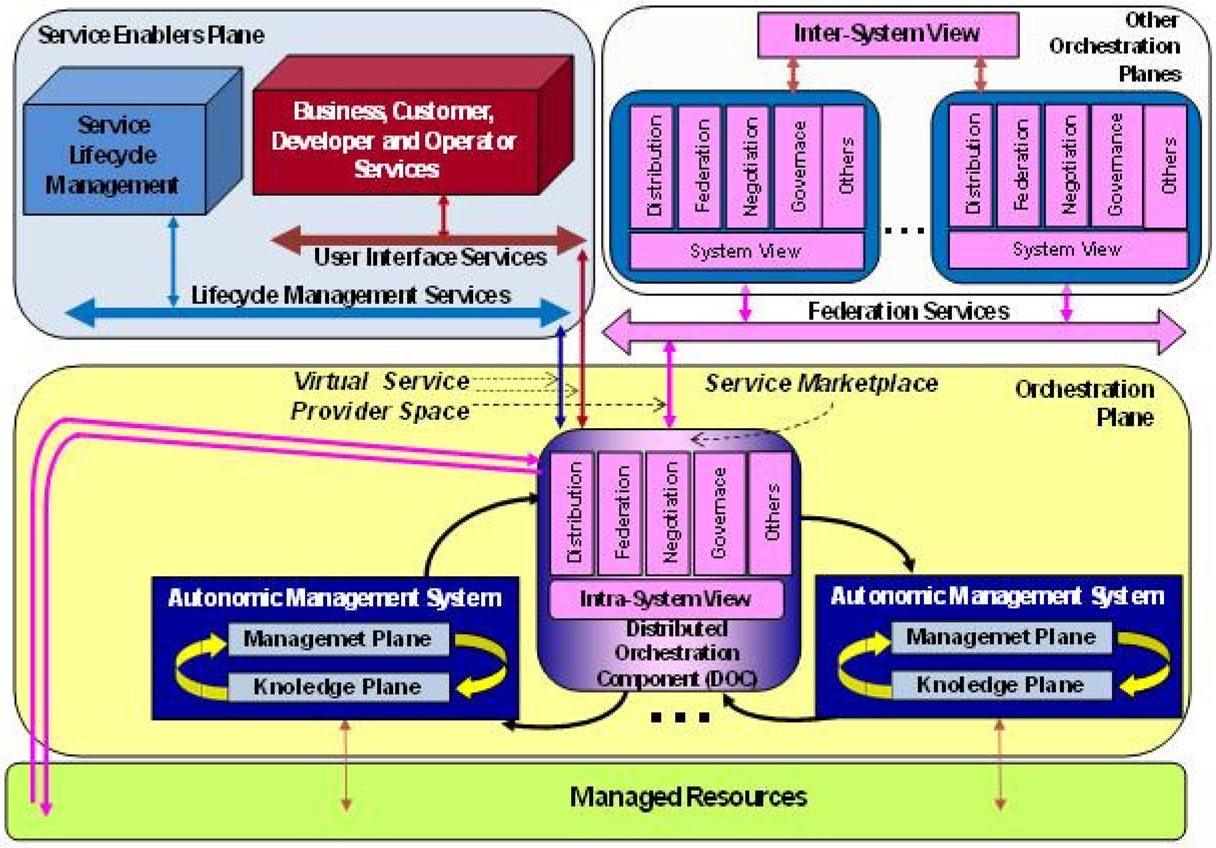


Figure 6-3. Simplified Autonomic Internet Orchestration Overlay.

The Orchestration Plane can be thought of as a control framework into which any number of components can be plugged into in order to achieve the required functionality. These components could have direct interworking with control algorithms, situated in the control plane of the Internet (i.e. to provide real time reaction), and interworking with other management functions (i.e. to provide near real time reaction).

The AutoI Orchestration Plane has key functionalities that advance the state of the art. Relevant for our study is the fact that the Orchestration Plane is made up of one or more Autonomic Management Systems (AMSs), one or more Distributed Orchestration Components (DOCs). Each AMS represents an administrative and/or organisational boundary that is responsible for managing a set of devices, sub-networks, or networks using a common set of policies and knowledge. A set of DOCs enable AMSs to communicate with each other. A set of buses enable the Orchestration Plane to be federated with other Orchestration Planes.

The Orchestration Plane acts as control workflow for all AMSs ensuring bootstrapping, initialisation, dynamic reconfiguration, adaptation and contextualisation, optimisation, organisation and closing down of AMSs. It also controls the sequence and conditions in which one AMS invokes other AMS in order to realise some useful function (i.e., an orchestration is the pattern of interactions between AMSs).

Central to our study is the fact that the AMSs are designed to be federated, enabling different AMSs that are dedicated to governing different types of devices, resources, and services to be combined. Each AMS advertises a set of capabilities that can be used by other AMSs; such functionality can be negotiated (e.g., through pre-defined agreements, auctioning, and other mechanisms).

An AMS collects appropriate monitoring information from the virtual and non-virtual devices and services that it is managing, and makes appropriate decisions for the resources and services that it governs, either by itself (if its governance mode is individual) or in collaboration with other AMSs (if its governance mode is distributed or collaborative).

Service-wise, the AutoI orchestration overlay can be seen as a Virtual Service Provider. Also its Distributed Orchestration Component (DOC) should support a service marketplace functionality so that the AMSs can negotiate and perform service coalition formations transparent to the service requester.

6.3 Description of the Service Coalitions Supporting Algorithm

6.3.1 General Features

The algorithm presented in this work enables Virtual Service Providers act as service brokers, transparent to the customer. More specifically, a VSP enables an electronic marketplace for service providers as the algorithm is embedded in the management systems of the service providers. Associated to a VSP we distinguish the following concepts:

- A *requester entity Rq* is the triggering entity in the Virtual Service Provider that starts the coalition formation as a service request.
- A *service task T* is a specialisation of the requested service, and is specified by a set of *k service skill*[1] requirements: $T = \{T^1, T^2,..., T^k\}$. The specialisation of this *service task T* is a function of the Virtual Service Provider. Each service skill T^j is a necessary subtask for performing the *service task T* (specialised requested service).
- *Provider entities* are the Service Providers that offer their services to the Service Marketplace for potential coalitions with other providers. Each provider is noted as σ_{ij} and is characterised with a specific set of service skills[1], and different performance levels for each service skill[1]: $\sigma_{il} = \{\sigma^1_{il}, \sigma^2_{il},\ldots, \sigma^k_{il}\}$.
- *Coalition of service providers* is the result of a marketplace service coalition formation process, namely it is the result of the execution of the algorithm. A coalition of service providers is noted as an array of members: $\sigma_i = \{\sigma_{i1}, \sigma_{i2},\ldots, \sigma_{im}\}$.

6.3.2 Main Features of the Algorithm

The algorithm has the following main features:

6.3.2.1 Skill Aggregation

In a service coalition, the *service skills* of the *provider entities* are aggregated in such a way that each provider gives the best of itself in a joint effort to create a group as competitive as possible under the requirements of the *service task*. For each service skill T^j the coalition has a value in which the aggregated efforts of its members are represented. In this manner, the provider in a coalition which is the best fit for

[1] The term *service skill* refers to the capabilities, functionalities, **services** or other support provided by a provider within the market place

performing a certain subtask would be the one that performs it: $\sigma_i^l = max(\sigma^l_{ij})$: $1 \leq j \leq m$. The aggregated effort of providers is used to measure a score $scr(\sigma_x, T) = \sum_{j=0}^{k} (\sigma_x^j * T^j)$ that indicates how well the providers in coalition σ_x perform together for accomplishing a *service task T*. This value is modelled as a regular scalar product that weights each service skill with the importance of each requirement. Other aggregation policies have been analysed and can be found in [107].

6.3.2.2 Iterative Process

The coalitional process by which the *service providers* work together in order to tackle the required *service task T* is not based on a one shot optimisation calculation, instead, the algorithm process unfolds in an iterated manner, creating coalitions that incrementally improve their quality based on providers self-interested decisions.

6.3.2.3 Self-Interested Service Provider's Behaviour

Providers are considered self-interested and their prime objective is to be in the most competitive coalition in order to be elected in the negotiation. To fulfill this generic objective, they follow a score maximising strategy [104] meaning that each provider evaluates its contribution to other existing coalition in which its score is higher than in its current coalition (or equal in score but smaller in coalition size). If by accepting this new provider's proposal, the hosting coalition also improves its current situation, the provider will leave its current coalition to join the new hosting one.

6.3.2.4 Limiting Communication amongst Providers

A large scale system allowing service providers be aware of every other service provider in the system can become prohibitive. To tackle this issue we have studied different forms of limiting the social awareness of providers, by letting them have a reduced and fixed set of connections that limits the coalitions each provider can evaluate at any time. More specifically, for an individual service provider σ_{ij}, a social network is a set of providers $V_{/} = \sigma_{kl}, \ldots, \sigma_{pq}$ for which σ_{ij} can consider its inclusion in their coalitions, formally: $SN\sigma_{ij} = V_{/}$. We have studied different topologies of social networks [108], [109] concluding on the general advantages of random social networks which in turn have been considered in the experimental validation of this algorithm.

6.3.3 Main Steps of the Algorithm

The following is a list of the main steps that are carried out during the execution of the algorithm proposed. The algorithm is executed each time a new service is needed:

6.3.3.1 Initialisation Steps

S.1 Update the number of service providers (σ_{ij}), and for each of them update the services and the quality that they will be able to guarantee.

S.2 Update/(re)define the social network that interconnects the service providers.

S.3 Update/(re)define the entity that will issue the service requests, namely the *Rq* that will monitor coalitions' proposals.

S.4 Handle the *service task T* that will be requested.

S.5 (Re)define the initial coalitions of just one element for each one of the providers.

6.3.3.2 For the Requesting Entity Rq

R.1 Wait upon reception of a service request from a client entity (for instance the Orchestration Plane of AutoI).

R.2 Start the algorithm communicating the requested service (*task T*) to every provider and allowing the negotiation to start.

R.3 While coalition values are not stable (the providers have not finished their negotiation process):

- Trigger randomly (in random order) the different providers to take a joining action. These actions are meant for a service provider to define the coalition formation that is willing to share.
- Monitor resultant coalition values after all the providers have been asked for an action and they have decided the most suitable coalition for them.

R.4 End the algorithm presenting the coalitions to the client entity.

6.3.3.3 For the Provider σ_{ij}:

P.1 Wait for the trigger coming from the *Rq entity* to submit a coalitional action.

P.2 Ask the *Rq entity* for an evaluation of the score of its current coalition.

P.3 Ask the *Rq entity* for an evaluation of the potential score and size of the coalition resultant of expelling some member of its current coalition.

P.4 Ask the *Rq entity* for an evaluation of the potential score of joining a coalition of some of its social neighbours ($SN\sigma_{ij}$).

P.5 Ask the *Rq entity* for an evaluation of the potential score of replacing one or more providers in a coalition of some of its social neighbours ($SN\sigma_{ij}$).

P.6 Perform the coalitional action that either, improves its current score, or that even when the score may remain equal the coalition size could be reduced. Only are eligible the options that improve the target coalition score or that reduce the coalition size when the score is equally comparable. If no action is possible, the provider stays in its current coalition and communicates to *Rq entity* its decision.

6.4 Functional Proof of Concept Scenario

This Section illustrates some of the above introduced concepts by means of an scenario with a small number of elements. Consider an orchestration environment in which ten federated AMSs provide three main services; video content distribution, end-to-end connectivity and encryption services. Each provider is capable of providing the above services with the following specific quality degree:

A1: High Quality Video Content; A2: Medium Quality Video Content; A3: High Quality Video Content; A4: Medium Quality Video Content; A5: EF Connectivity Service [TT1, TT2, TT4]; A6: AF Connectivity Service [TT1, TT3, TT5]; A7: BE Connectivity Service [TT1, TT2, TT4]; A8: EF Connectivity Service [TT1, TT4, TT5]; A9: AF Connectivity Service [TT1, TT2, TT5]; A10: Encryption Service [TT1]

Consider a service requirement for the Orchestration Overlay as to "*deliver video content from/to ingress/egress points of traffic trunk TT1*" for which a service coalition formation is needed. At some point of the Orchestration's Service Lifecycle management support, the Distribution Orchestration Component (DOC) function will determine that there are two services involved in such requirement, the video content and the connectivity service between ingress/egress of Traffic Trunk 1 (TT1), each one with the same relative degree of importance. According to the algorithm procedure described earlier, the requirements definition are modelled as an array of individual requirement levels that would contain the following fields:

Service task T = [Video Content requirement level, E2E Connectivity Service requirement Level, Encryption Service requirement Level]

For this concrete service requirement, the task to fulfil will be defined as: *Service task T1 = [1, 1, 0*]. Likewise, service providers are also defined as an array of individual capabilities. For this specific scene service providers are modelled as:

P1=[10,0,0], P2=[5,0,0], P3=[10,0,0], P4=[5,0,0], P5=[0,10,0], P6=[0,5,0], P7=[0,1,0], P8=[0,10,0], P9=[0,5,0], P10=[0,0,10]

For this scenario we have created a sparse random social network between these service providers as shown in Figure 6-4 step 0. The social network here has a total of 16 edges (bi-directional connection sight between service providers). Each provider occupies a node in the network, and each provider has from two to four connections to other providers.

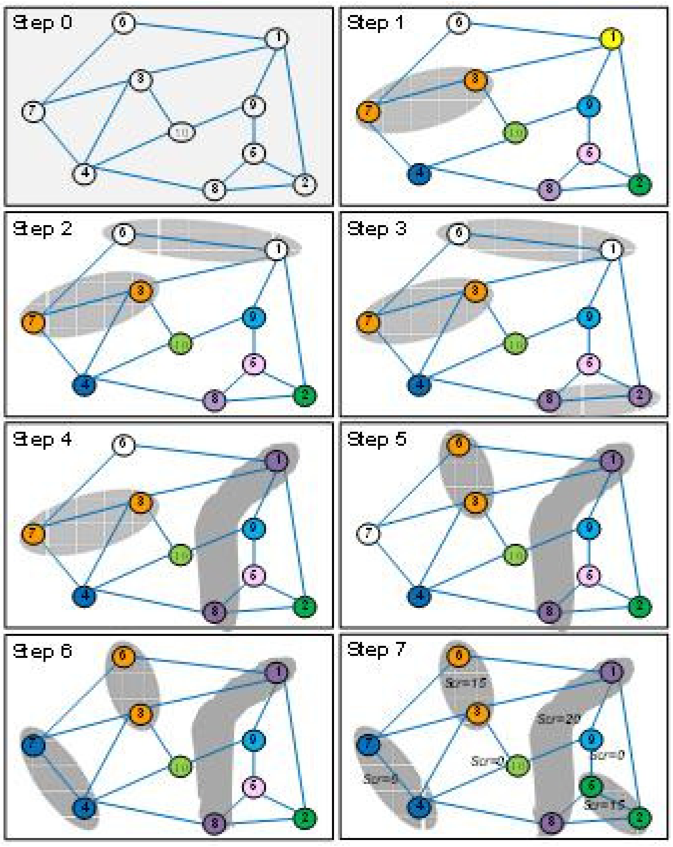


Figure 6-4. Partial coalition formation process.

Departing from an isolated state (every provider alone in a coalition of one member – see Figure 6-4 step 0), the process converges after 2 iterations into a steady state where none of the providers can improve its situation any more. The best coalition at this state corresponds with one of the possible solutions that maximise the coalition score. Let us see the evolution of the social networks as the algorithm progresses. The specific decisions taken by the providers during the first iteration involve 3 coalitional changes: (i) provider 3 creates a joint coalition with provider 7 (Figure 6-4 step 1); (ii) provider 1 creates a joint coalition with provider 6 (Figure 6-4 step 2); (iii) provider 2 creates a joint coalition with provider 8 (Figure 6-4 step 3).

During the second iteration the decisions taken by the providers involve four coalitional changes: (i) provider 1 leaves its previous coalition to replace provider 2 in its coalition with provider 8 (Figure 6-4 step 4). Note that at this step provider 1 is in a coalition with one with which it does not have a direct social connection; (ii) provider 6 that was abandoned by 1 in the previous iteration, now replaces provider 7 in its coalition with provider 3 (Figure 6-4 step 5). These two service providers are also bringing efforts together without being neighbours in the social network; (iii) provider 7 creates a joint coalition with provider 4 (Figure 6-4 step 6); (iv) provider 5 creates a joint coalition with provider 2 (Figure 6-4 step 7).

The role of the service requester can be adopted by any coalition, giving highest priority to those with higher score. In our scenario the coalition between provider 1 and 8 has the higher score (Scr = 20 – Figure 6-4 step 7) once the algorithm is executed. If the requester should be the best coalition, this one would be selected. However, if in the current example the requester needs two proposals for example, he would pick the best two coalitions. However, the second and third coalitions have exactly the same score (Scr = 15 – Figure 6-4 step 7). Having the same score means that from the requester point of view, they are indistinguishable, and the ranking order between them is random. In the example, after the algorithm execution, the resultant second scored coalition was the alliance between providers {6, 3} – coalition between providers {5, 2} has the same score (Scr = 15).

It is worth mentioning that the algorithm contemplates that each provider just evaluates (computes) a fixed and limited set of coalition options driven by its direct neighbours in the social network. This way for example, the coalition between provider 6 and 3 is possible (see Figure 6-4 step 5) given that provider 7 is connected to provider 6 (in its social network) and that providers 7 and 3 had previously formed a coalition. Having provider 6 joined the coalition between providers 7 and 3, the effect of algorithm's P.6 procedure is that provider 6 and provider 3 self-interestedly will form a coalition.

6.5 Experimental Validation in a Large Scale Scenario

One of the most valuable characteristics of the proposed algorithm for its applicability in FINs-oriented environments is its scalability. The mechanism can deal with very large populations of providers by constraining the connections that each one of them can have. However, this connectivity limitation can compromise the optimality of results obtained if the social network is too small. In this section the effect of using different sizes of social networks for a larger scale system is analysed in a specific set of scenarios.

We initially validate the coalitional results considering the social network size, demonstrating that in an environment of a reasonable size, even with small social networks, the results obtained are near optimal. The parameters of this experiment are the following:

- A requested complex service task composed of 10 different services, and different degrees of requirements for them:

  service task T1 = [5, 8, 11, 8, 3, 2, 10, 1, 1, 1].

- A population of 100 different providers: each provider has a certain performance value in 10 different services. The sum of the performance values in all the services in every provider is 200 and the standard deviation ranges from 35 to 40. A maximum value of 100 has been imposed in the performance of a service. This high standard deviation distribution of skills creates a setup in which all the agents are specialists in the provision of a small number of services, and also capable of providing some other services but with low performance levels. For example, providers 98, 99, and 100 are defined as:

  σi98 = [0, 0, 0, 0, 88, 0, 15, 0, 97, 0] σi99 = [0, 93, 0, 91, 0, 0, 1, 8, 0, 7] σi100 = [0, 0, 0, 100, 90, 0, 0, 0, 10, 0]

- A total of 3500 different social networks grouped in seven sets with random topology are considered. Each network has 100 nodes and a number of connections (edges) ranging from 100 to 500 according to the set it pertains (see Table 6-1 for a description of the network characteristics). Each experiment consists in running the coalition algorithm for each network of each set and getting the corresponding averages over the set. The provider for each network is assigned randomly to one of the nodes. To create the networks, an algorithm based on [110] has been used. This has the property that the number of connections of each node (degree) is relatively similar for all the nodes. Figure 6-5 show two social networks with 100 and 500 edges according to the characteristics shown in Table 6-1.

Table 6-1. Social network characteristics.

setname	numberofnets	numberofedges	avg.degree	avg.pathlength
A	500	100	2	10.74
B	500	150	3	4.34
C	500	200	4	3.45
D	500	250	5	3.03
E	500	300	6	2.73
F	500	400	6	2.41
G	500	500	6	2.21

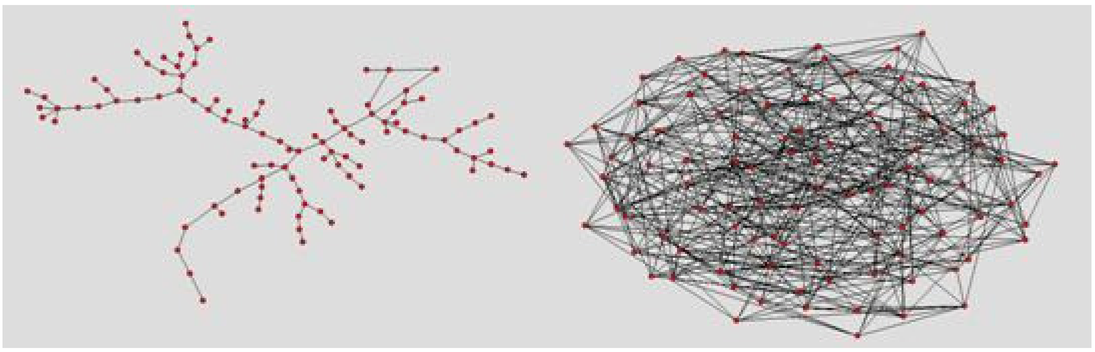


Figure 6-5. Social network topologies from Table 6-1: left – set A, right – set G.

In order to obtain the score of the best possible coalition, the algorithm has been run using a fully connected social network. The score obtained (Scr = 5000) was then compared to the best coalition's score in each experiment for each series of 500 networks. The percentage of results that coincide with the optimal score is showed in Figure 6-6. Experiments reveal that from an average degree of 4 (series C) onwards, the accuracy of results is above 70%, and that in the experiments with average degree

= 6 and average path length of the social network = 2.21, all the results obtained equated the optimal score obtained with the fully connected network.

Using a sparsely connected network involves a significantly reduction in computing time. Computation time is directly related with the amount of evaluations that service providers perform to decide which coalition is better for them. Figure 6-7 shows the average number of times that service providers evaluate a coalition. In this graphic, the direct relationship between the density of the networks and the evaluations performed is shown.

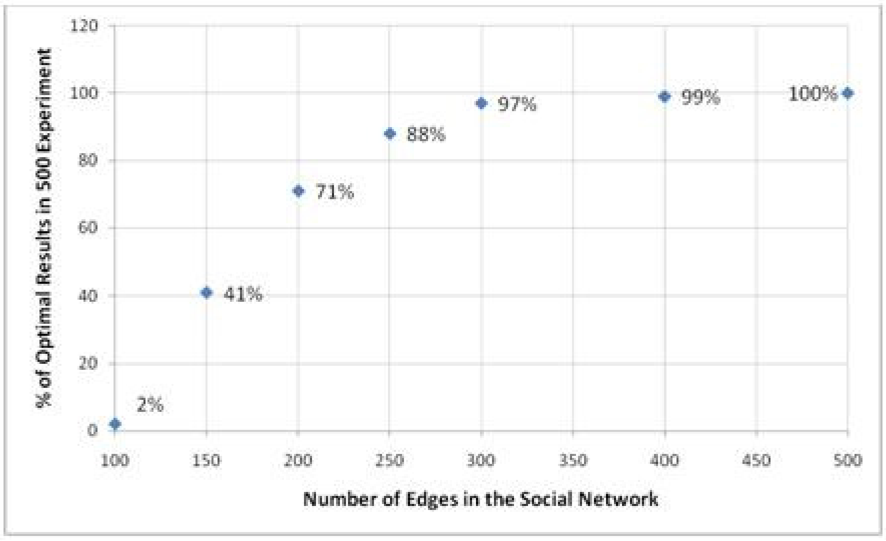


Figure 6-6. Percentage of coalitions that provide optimal results.

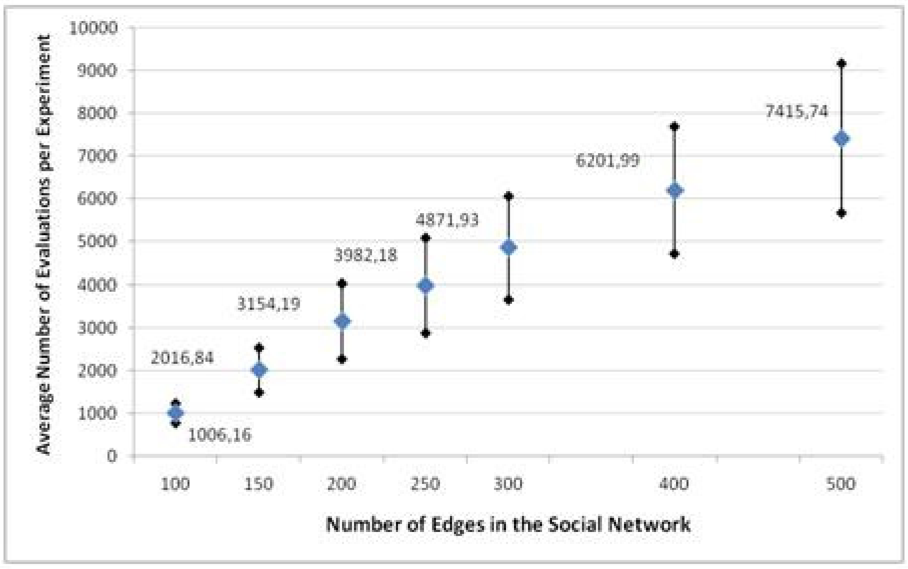


Figure 6-7. Average number of evaluations performed to generate a solution for the different social networks series. Standard deviation is also represented to validate the statistical significance of result.

In order to consider the deployment of the algorithm in a system, the obvious preferred option is to reduce computation time and increase the probability of obtaining the optimal solution, however, it exists an inverse relationship between these two variables.

Hence it would be necessary to find the right balance between the acceptable computation time and the required accuracy. It is important to note that the algorithm presented could be optimised in many different manners in order to reduce the number of evaluations, for example parallelising evaluations of providers for which their possible actions do not interfere, or by time-stamping coalitions in order not to re-evaluate options that have already been evaluated. Nevertheless, this initial study is focused in the analysis of the feasibility of the coalition formation mechanism in service composition for FINs, hence the optimisation possibilities of the algorithm are out of the scope of this report.

6.6 Conclusions

We have presented an algorithm that will potentially be used to set up electronic marketplaces for service providers. In order for this to happen, the management systems involved in the service negotiations must have management systems in charge of the negotiation process. One of the target application domains that we have identified for this is in the provisioning of Future Internet Services for which a strong emphasis on service providers' collaborations has been given in the later years. We have demonstrated how the algorithm provides support for the service marketplace coalition formation process in such environments and have evaluated its scalability and the conditions under which the algorithm converges to optimal solutions.

The algorithm presented in this report has been evaluated with fixed social network topologies. Even when the algorithm is valid for any network topology, the effect of the dynamicity of social networks on the amount of evaluations and on the optimality in the resulting coalitions has not been explicitly addressed and will be part of our future work. In addition, future work will be directed to extend and evaluate the algorithm under multiple simultaneous service requests. Aligned to this study the dynamic load balance feature will be subject of our study.

We plan to implement this algorithm in the Autonomic Internet architectural model for which the studies presented in FeMaFI have provided some hints of how such a futuristic approach may be put in practice with regard to the service marketplace environments.

7 P2P Collaboration and Storage (P2P COST)

7.1 Introduction

Storing large amounts of data in a persistent manner usually comes at a high cost in terms of hardware infrastructure and support services if done based on centralised servers. Moreover, such centralised storage systems are subject to server crashes, network outages, and their extensibility is typically limited.

At the same time, emerging P2P storage networks are able to provide highly-available access to huge amounts of data by leveraging the resources of multiple inexpensive and untrusted storage devices distributed over a wide area.

However, some problems are currently still unsolved. Incentives for storage provider and consumer must be in place for the network to provide a satisfactory quality of service. Also, high performance bandwidth requirements in P2P storage networks are not always met. A globally unique identifier in a P2P storage network makes changes difficult to be tracked as most P2P storage systems use a hash value generated based on the content. A small change in the content leads to a totally different hash. Moreover, a P2P storage system needs to cope with malicious peers that can degrade performance and reliability. Finally, in a fully distributed P2P storage system, there is no access control and administrative methods in place.

The P2P COST activity has been split in several subtasks. First, several existing P2P storage networks have been compared in terms of features, efficiency, and throughput. DRFS – a generic P2P file system has been designed, implemented, and evaluated as described in Section 7.2. In order to provide improved security in a heterogeneous environment, incentive mechanisms need to be in place. Section 7.3 shows an incentive model to prevent hidden actions in a P2P storage system. Another important and difficult task in any P2P network is to protect against the Sybil Attack. Section 7.4 evaluates different methods to accomplish this task. Finally, Section 7.5 models a search engine for structured P2P storage systems and studies the specific case of a P2P bug tracking system.

7.2 DRFS – Distributed Reliable File System

Distributed file systems provide access to data in a networked environment. If such systems operate in a client-server (C/S) mode, e.g., NFS (Network File System) [111] or SMB (Server Message Block) [112], issues concerning scalability, presence of a single point of failure, and fault tolerance emerge. Scalability issues, such as coping with an increasing number of clients, need to be addressed, since bandwidth on the server side may be limited and expensive. In case of server failure, a total lack of service follows; for distributed file systems, all data stored on a server becomes unavailable. A self-organising, fault-tolerant infrastructure requires less management from the service provider, thus reducing cost [113] and down times.

Peer-to-peer (P2P) systems [114], [115], [116], [117] are used to design scalable, fault tolerant distributed systems providing high availability of shared resources. Distributed file systems based on P2P networks can help to avoid the problems of a C/S architecture.

The case considered is a small to medium sized office LAN where trust among participants is assumed. Each workstation running DRFS software dedicates disk space for DRFS resulting in a transparently accessible shared file system. Such partition could be used as a media repository, where media files are published and accessed by many employees. Since it is a cooperative network, presence of hostile peers are not expected.

To address all those aspects, the Distributed Reliable File System (DRFS) has been designed and a prototype has been developed and evaluated. DRFS shows the major characteristics of a cooperative, read-write, distributed and decentralised P2P file system that can tolerate node failure without data loss. To prevent data loss due to peer failures, replication is used. Access to DRFS is done throughout a FUSE (Filesystem in Userspace) interface, which allows users and applications to access data transparently. Evaluations show its scalability and reliability with a low overhead.

7.2.1 Related Work

A categorisation of P2P file systems is provided by Hasan et al. [118]. These surveyed P2P file systems usually concentrate on some particular problems. Some systems do not support the update of existing information (write) while others support write operations under certain conditions, some of them use special interfaces to access the stored data, and in others the location of the data is correlated to its content. Thus, the following related work focus on the following key dimensions which are relevant for DRFS: read/write support, access interface, and data storage mechanism. Pure file sharing applications are not relevant for DRFS because persistence of non-popular data storage is not guaranteed.

The Cooperative File System (CFS) [119] is a read only P2P storage system. It stores blocks and spreads blocks evenly among the peers. The system consists of three distinct layers: a file system client, a distributed hash table (DHT) layer, DHash, and a Chord [120] layer, used for lookup and routing. Files on the CFS system are accessed trough the file system client, which interprets the blocks as files. Normal CFS peers can only read the stored data, while publisher may publish new data. Data expires and is lost after an agreed-upon time interval. If the publishers want the data to persist, they must request a time extension from the CFS system. To delete a given data, a time extension is not requested for it and it is left to expire.

PAST [121] is a large-scale P2P persistent storage system. It operates on top of the Pastry [115] lookup system. PAST semantics diverges from that of a general purpose file system: facilities for searching, directory lookup and traversal are not provided. The operations provided by the software client are insertion, lookup and reclaim. PAST stores entire files on its peers. When a file is stored it is associated with an unique identifier and cannot be updated anymore. PAST uses replica diversion and file diversion. If a peer designed to store a file doesn't have sufficient space at its disposal, then the replica diversion mechanism saves it on a neighbouring peer, while the first peer only stores a pointer to the actual location. However, if the replicas still cannot be diverted within a leaf set, the file diversion mechanism generates a new identifier.

IVY [122] is a log-based, distributed and decentralised read/write P2P file system. It is able to support multiple users concurrently. IVY's goal is to provide integrity even when participants don't fully trust each other or the underlying storage system. In IVY, each participant keeps a log of changes they individually made to the file system. These logs are stored in the DHash distributed hash table. When a peer issues a file data or meta-data lookup, it scans all the logs associated with the file. The current global state of the file system is a composite of all logs. Since every modification can be identified, actions of misbehaving peers can be undone and discarded. Multiple peers are allowed to write simultaneously to a resource since they write in own, separate logs. However, resources may be in an inconsistent state. While IVY can provide some information about the inconsistencies, explicit conflict resolution tools have to be used.

OceanStore [116] provides a large scale, persistent storage facility for alterable data objects and reliability in a constantly changing environment. Data privacy and integrity are achieved trough encryption and a Byzantine agreement protocol is used to update the replicas. Pond [123], the current OceanStore prototype, uses Tapestry [124] routing overlay mechanism to store blocks of data at nodes and for lookup and routing. Each object stored is represented as an ordered sequence of its versions, that is for each time it has been updated. To update an object, a consensus of nodes holding information about the object is needed. The replication of data blocks is based on erasure coding.

Kelips [125] is a read-only P2P file system built to achieve constant look-up times while being resistant to churn and failures. These improvements come at the cost of increased memory utilisation and continuous background communication overhead. Precisely, it can achieve a lookup time of $O(1)$, while the memory usage is $O(n)$, where n is the number of nodes in the system. Files are stored entirely and cannot be updated. The access interface to the Kelips P2P file system is not known.

Table 7-1. Comparison of different P2P file systems.

	Read/Write support	Access interface	Data Stored
CFS	Read only	UNIX-like interface	Blocks addressed by content key
PAST	Read only	PAST client	Entire Files
Ivy	Read/Write	UNIX-like interface	Logs
OceanStore	Read/Write	App. dependent	Erasure code fragments
Kelips	Read only	Unknown	Entire files
IgorFS	Read/Write	FUSE	Blocks addressed by content key
DRFS	Read/Write	FUSE	Blocks addressed by random keys

IgorFS [126] is a distributed P2P file system built on top of a Chord-like structured overlay. IgorFS allows transparent access to remotely stored data trough FUSE interface. Files stored are split in blocks of a given size. After the splitting, the blocks are processed for encryption in the following way. The block data is initially hashed once, with value k as the result of the hash. The value k is then used as a key to encrypt the data of the block. Once the block is encrypted, it is hashed again and the result is used as the identifier, under which the encrypted block will be stored in the DHT. A file is represented as a collection of (offset, ID, k) - tuples, and directories are represented as collection of file names and their attributes. This meta-data is serialised, split in blocks and encrypted, just like data blocks. If a user obtains an (ID, k) pair for a directory block, the user can decrypt the directory block and get the content. Successively, the user can get access to all the subdirectories and their files. Whenever a content of a single file changes, the relevant blocks must be re-hashed, re-encrypted and eventually moved in the DHT because the newly obtained ID could indicate another node. These changes must also be done in the meta-data, and eventually be propagated to the root of the file system. Since this causes computational overhead, the publishing frequency is restricted and the modifications to a file system become permanent after a snapshot. Table 7-1compares related work using the dimension read/write support, which describes which operations are available in a given file system. Access interface indicates the mechanisms used by the file systems to allow access to stored data. Data stored indicates how the data is actually stored and how it can be addressed.

From the reviewed P2P file systems, only IVY, OceanStore and IgorFS support write operations and update to files. In IVY, concurrent modifications to a resource can produce inconsistent resource states, which have to be explicitly dealt with. In IgorFS, resources are saved and addressed accordingly to their content-hashes. When update occurs in a single file, an entire branch of the file system tree might have to be recoded. Therefore, updates to the files are not visible immediately: they become permanent only after a snapshot. OceanStore uses erasure coding to deal with replicas, but while it has been proved that erasure coding performs better than replication in terms of storage space, it also brings an extra computational overhead and complexity in the system [127], [128]. Ivy and OceanStore are designed to operate in an insecure environment. However, this also brings some drawbacks: in Ivy each action of every peer is logged and stored forever, impacting on storage capacity; in OceanStore, use of the Byzantine agreement protocol puts an additional toll on the system's performance. PAST and Kelips do not offer a standardised access interface to data. Both systems store entire files rather than file blocks, which requires more complex storage balancing schemes, such as replica diversion and file diversion in PAST.

7.2.2 DRFS Design and Implementation

DRFS is designed to provide a persistent, reliable, self-managed and distributed file system in a cooperative environment. A typical scenario is a small to medium-size company or department, or a home network, in which spare hard disk space is used to build a reliable storage for important data.

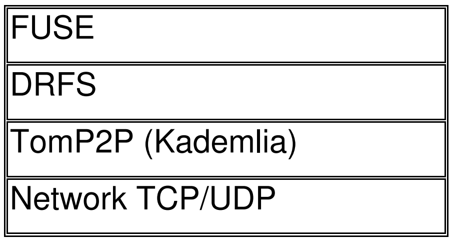


Figure 7-1. DRFS Architecture.

The DRFS architecture uses a layered architecture, as shown in Figure 7-1. At the network level, UDP is used for messages that fit in a single packet; otherwise TCP is used due to its flow and congestion controls. On top of the network layer, the TomP2P [129] overlay network is used. TomP2P is a Kademlia [117]-based, P2P framework, developed at University of Zurich. Besides distributed hash table (DHT) operations such as *put* and *get*, distributed multi map (DMM) operations such as *add*, *remove*, *size*, and *contains* are available in TomP2P. Additionally, each operation supports a *domain* parameter to avoid key collisions, *e.g.*, *put(0x123, "path",value1)* that stores *value1* under the key *0x123* and *put (0x123, "attribute", value2)* that stores *value2* under the same key do not collide and can be retrieved using *get(0x123, "path")*, respectively *get(0x123, "attribute")*.

DRFS implements the FUSE [130] interface to mount it as a regular file system in UNIX environments. It supports the creation of directories, files and UNIX-like symbolic links. Like in the UNIX file system [131], metadata (name, attributes) is separated from the content.

7.2.2.1 Assumptions

The following assumptions for the DRFS design are used. To guarantee the persistence of all stored data, it is assumed that the hosts in such an environment have a high online time. Work in [132] shows that, assuming mean peer availability 80%, using a replication factor of 5 yields data availability of 0.9997. Furthermore, trust among DRFS participants is assumed. Thus, DRFS deals with node failures but not with malicious behaviour. A third assumption is that nodes can concurrently access data.

7.2.2.2 DRFS P2P File Systems Design

File and directory descriptors are stored in the DHT under the key of their parent directory's path name, except for the root directory, which is stored under a predefined key. A directory descriptor contains a collection of file and directory descriptors, allowing a hierarchical structure. Using of these collections exploits the *add* feature of the DMM.

A file descriptor contains references to file fragments by storing the key under which the fragment can be found. File content is split in fragments, which allows a better load balance because fragments are distributed randomly in the DHT to avoid that a small number of nodes is responsible for large files. File fragments also allow faster access to specific portions of a file, because only the fragments containing the needed data are fetched. File fragments are saved in the DHT by computing random 160-bit keys and a combination of path name and fragment number as *domain*. Random keys may collide but the *domain* will always be different because path names are unique, thus, no data will be overwritten due to key collisions. An example is shown in Figure 7-2, with a node 0x412 that stores the file descriptor with references to fragment 1 with random key 0x422 and fragment 2 with random key 0x132. These fragments are stored on those nodes closest to their IDs.

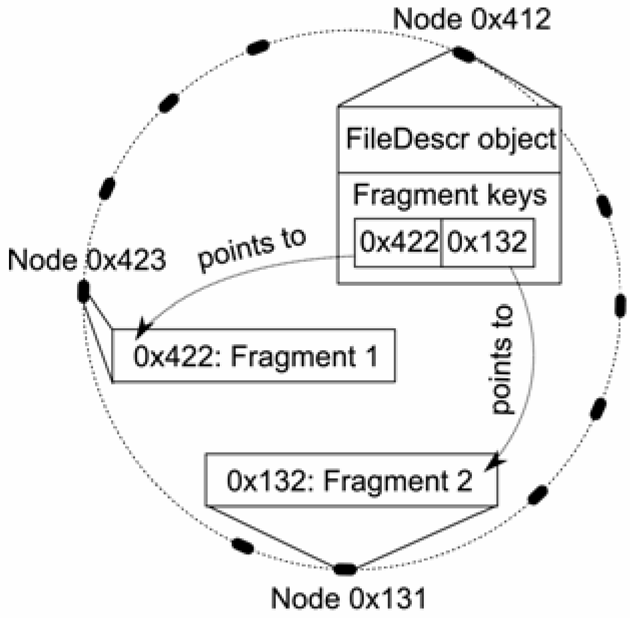


Figure 7-2. Example of file fragment storage in the P2P overlay.

A key feature of DRFS is that keys of file fragments are unaffected by its content. Updating a given fragment's data will not affect its naming scheme nor will invalidate the key under which the fragment is stored, thus, fragments remain on the same node. On the other hand, updating path names in a file and directory descriptor causes its hash to change, which may require the file and directory descriptor to be transferred to a different peer. However, the size of file and directory data is usually small because it only contains references.

Accessing a file requires one lookup in the DHT for the corresponding file descriptor because the key of the file descriptor is based on the path. The file descriptor contains a collection of file fragments. These fragments need to be looked in the DHT and retrieved. If a complete file is retrieved, all file fragments need to be retrieved.

7.2.2.3 P2P File System Replication

DRFS has an active replication policy. Nodes periodically scan their stored values and check whether the DHT contains enough replicas of the data by using the *size* operation. In case of insufficient number of copies, the replication process is started. Nodes that were offline during a deletion of a file and still contain a copy, might replicate the removed file or part of it. For this reason, deleted files are marked as deleted, but their resource descriptor resides in the DHT for a given amount of time, so that parts of that file are not replicated again. This technique is similar to that of using death certificates as described in [133].

DRFS uses the caching mechanisms from TomP2P. Whenever a node requests a file and directory description or a file fragment, it will remain stored for some time at the requesting node. This leads to time and bandwidth savings in case a node repeatedly requests the same key. Furthermore, availability is improved, since caching nodes become a source for other nodes and popular files will have higher number of replicas.

7.2.3 Evaluation

Performance of the DRFS system was measured by using goodput, i.e. the application level throughput, which corresponds to file size divided by transfer time. Other evaluated dimensions were the scalability of concurrent read operations with respect to the number of peers and the overall system overhead regarding different file sizes.

7.2.3.1 Typical File Size

Work in [134] reports a study of distributions of files in file, comparing them with statistics from older studies. It states that typically most bytes in a file system are found in large files. The mean file size reported, depending on the file system, is in a range from 169 KB to 29 MB, 80% of the studied file systems showing a mean file size that varies from 5.1 MB to 29 MB. The study also showed a shift in trend in statistics from older studies towards bigger file size: for example, it was found in [135] that the average file size is 0.079 MB.

7.2.3.2 Usage Patterns

Regarding usage patterns, it is assumed that the read operation is more common than the write operation. In [132] is stated that about 80% of file access is for reading. For this reason, the measurements are mostly for read operations. Another assumption is that the probability of multiple participants concurrently accessing the DRFS file system is high.

7.2.3.3 Testing Environment

The tests were performed on a 100 Mbit/s Ethernet network consisting of up to 16 hosts, belonging to 9 different subnets. All hosts had a similar configuration: Intel(R) Pentium(R) 4 CPU 2.8 GHz with 512 MB RAM running Debian Linux. Java virtual machine used was Java SE Runtime Environment (JRE) 6 Update 10. In tests in which DRFS performance was measured against NFS, the NFS server and client software used were version 4. To better illustrate the real data transfer performance, caching mechanisms were turned off and active data replication feature was also deactivated in all tests. Fragment size for DRFS was set to 1 MB, a value in the range of the mean file sizes.

7.2.3.4 R/W Performance Measurements

In this test, DRFS was tested with a number of peers ranging from 2 to 16. For each number of peers, a single node wrote a 100 MB file to the DRFS file system and then read the same file from DRFS. The time needed to complete each operation was recorded.

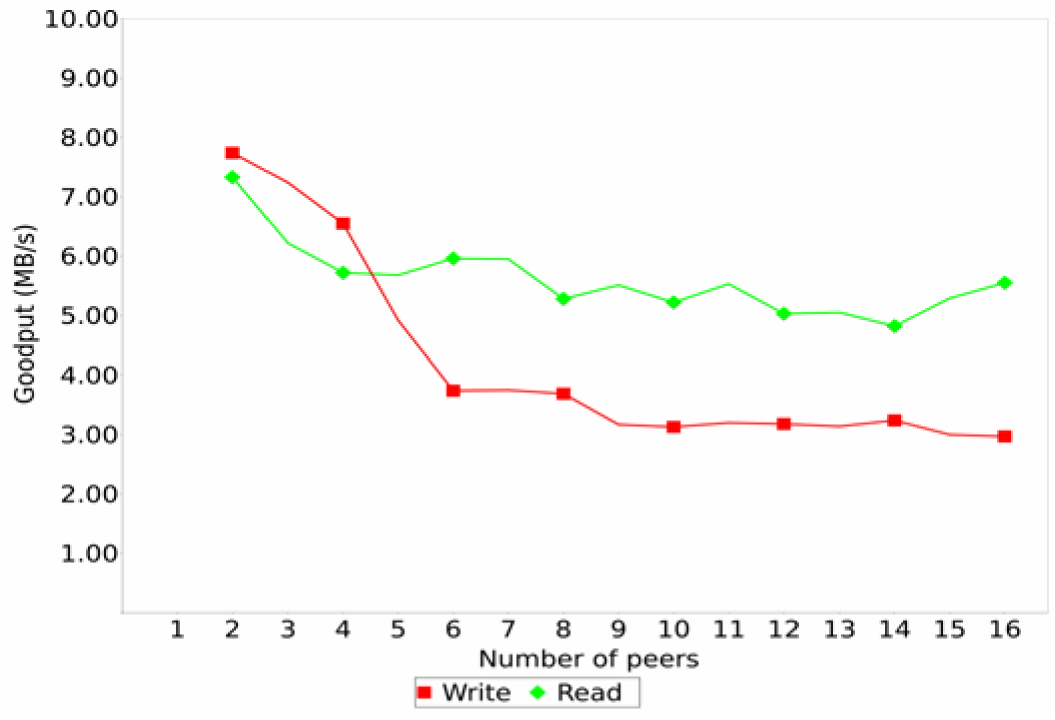


Figure 7-3. Read and write performance with increasing peer number.

Figure 7-3 shows the relation between numbers of peers in the DRFS system and goodput achieved for read and write. Each operation was repeated 10 times, and the average goodput is shown. For writes, the goodput drops from 2 to 6 nodes. This is due to the replication factor and the writing node concurrently sends data to multiple nodes (by default 5). For systems having more than 6 nodes the write goodput levels off. For the read operation, the goodput is highest with 2 nodes. This is because the overhead for fragment lookup is lower. If there are more peers than replicas, the read goodput remains stable. This measurement confirms the scalability of the operations read and write with regard to the number of nodes in the system.

7.2.3.5 DRFS vs. NFS Parallel Read

In the following experiment, two separate measurements took place. The first consisted of setting a NFS server storing a 100 MB file. Then a number of hosts (clients) downloaded the file from the server at the same time. The number of hosts was gradually increased up to 16. The time needed to download the file was measured at each node and, for each number of clients an average time per node was computed. The second measurement was performed in a similar matter, but using DRFS. Instead of being stored on a single computer, the file was saved in a fully distributed fashion in the DRFS system, with 16 nodes. Nodes concurrently copied the file from the DRFS system to their local storage. In both cases, the time was measured and averaged.

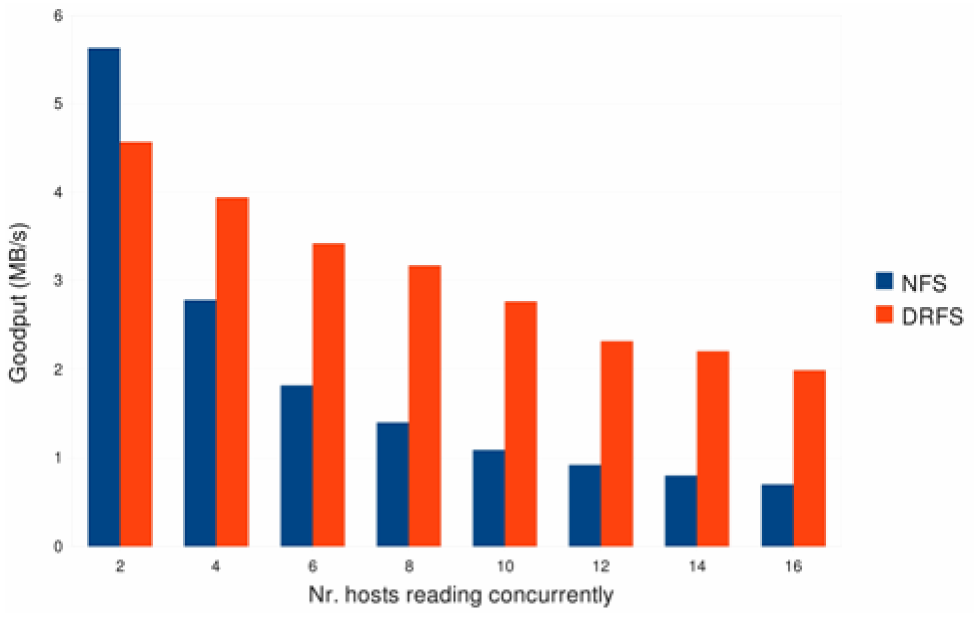


Figure 7-4. Concurrent read performance comparison between NFS and DRFS.

Figure 7-4 shows goodput as a function of the number of nodes downloading the same file in parallel, for both NFS and DRFS. It can be seen that, for NFS, the goodput drops with increasing number of nodes that concurrently copy the file. Obtained results generally conform to the function *G/n*, where *G* is the highest possible goodput and *n* the number of nodes downloading the file in parallel. The measurements show that DRFS performs worse than NFS only when 2 or less nodes copy the file at the same time. As the number of nodes is higher than 2, the nodes download the file faster with DRFS than with NFS. With 16 nodes, DRFS is 284% faster than NFS. This experiment shows that NFS is less scalable than DRFS.

7.2.3.6 Read Performance

In this experiment, a variable number of nodes read files of a given file size in parallel. File sizes used were in a range of 2^{10} to 2^{28} bytes (from 1 KB to 256 MB), doubling for each round of the experiment. The goal of this experiment was to identify the relation between file size, goodput, and the number of nodes concurrently copying the file. For each number of peers, the files were copied multiple times, to obtain an average. Since this measurement was time-consuming, the smallest file (1 KB) was copied 80 times, and for every increasing magnitude of file size, two less copies were made. Times required to copy the files were measured and averaged for each number of peers.

Figure 7-5 shows the average read performance for different file sizes when the number of parallel reads is 4, 6, 8 and 10. It can be seen that with small files (< 1 MB) the goodput achieved is the smallest in all cases. This can be explained if the steps necessary to copy the file are taken into account: hash the filename, fetch the file descriptor, read the location of file fragment from file descriptor and finally fetch the file fragment. For small files the actual data transfer is short compared to all other steps. However, when computing goodput, all these steps are included, hence the goodput is low.

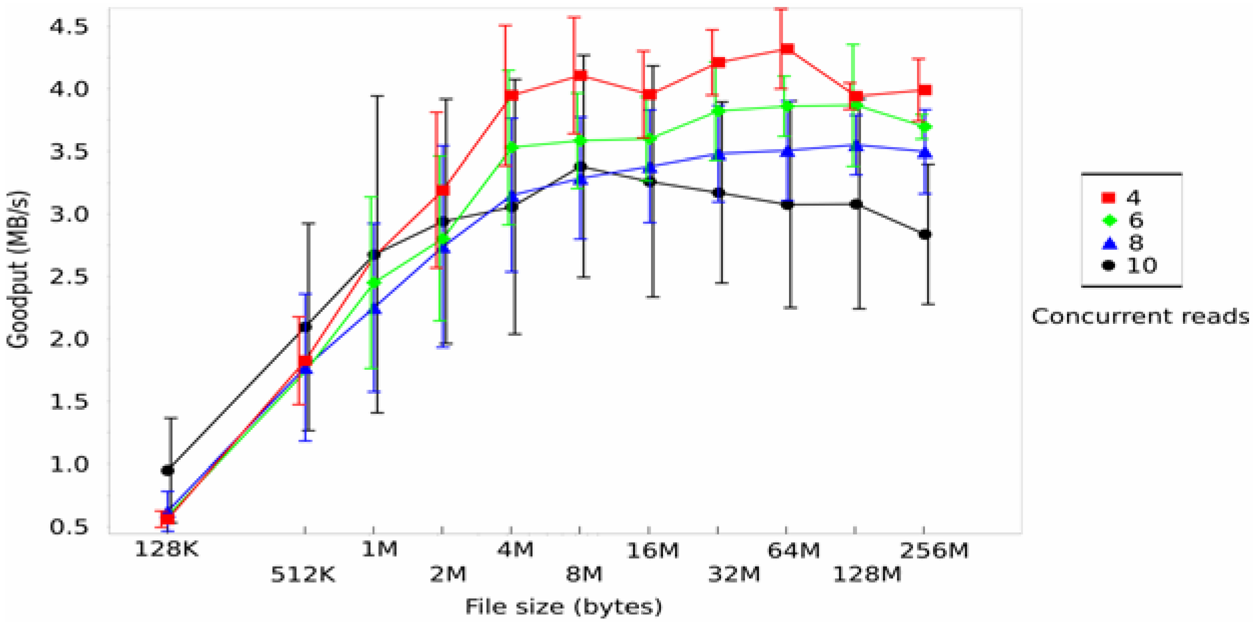


Figure 7-5. Concurrent read performance for different file sizes.

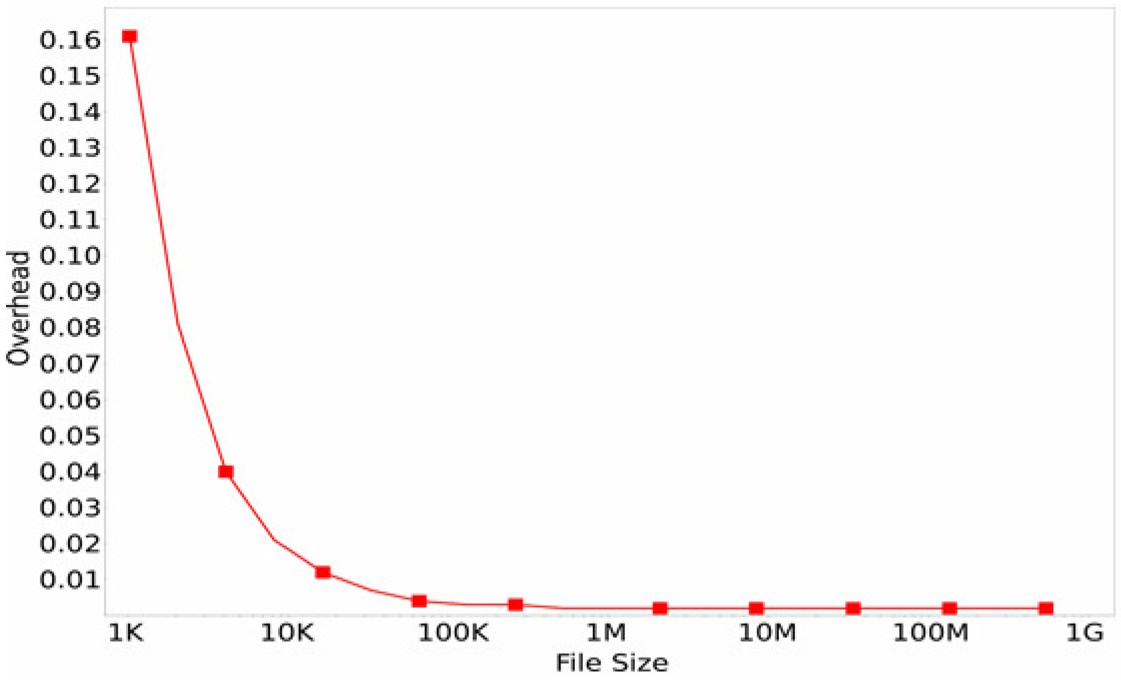


Figure 7-6. Overhead with respect to different file sizes.

For larger files, the performance is higher and there seems to be a clear trend especially for files greater or equal to 4 MB. In case of such large files a small performance drop can be observed when the number of nodes concurrently downloading the file increases. The goodput decrease is not directly proportional to the increase of downloading nodes, so that *e.g.* for the 4 MB file, when going from 4 to 8 nodes, the goodput lowers from 3.87 MB/s to 3.04 MB/s, that is 21.45% performance drop. This goodput decrease is due to congestion: since many nodes try to download the same file, and thus the same fragments, at the same time, they end up overburdening nodes that store the specific fragment. This experiment shows that DRFS performs well for medium sized and large files, and that it scales substantially with respect to the number of concurrently downloading nodes.

7.2.3.7 Other Measurements

For this experiment, a DRFS system with a fixed number of 10 peers was used. On each peer the application traffic was logged. Files of different sizes (from 1 KB to 512 MB) were stored in the DRFS file system multiple times. The overhead is the additional data sent over the network that is not related to file data and its replicas. Such overhead traffic includes P2P maintenance messages, routing messages, or storage and retrieval of file descriptors (metadata) messages.

Figure 7-6 illustrates the application traffic overhead. It can be observed that files less than 16 KB long have the highest overhead (16%). This was expected because the resource descriptor (meta-data) is in the order of kilobytes, thus creating a relatively high overhead. For larger files, the overhead is constant at 0.2%.

7.2.4 Conclusions and Future Work

DRFS is a distributed P2P file system that allows transparent read and write access to the storage by implementing the FUSE interface. It provides higher data availability through replication of data and higher fault tolerance through decentralisation.

All experimental results show that DRFS is scalable and the performance of read/write operations is not influenced by the size of the system. These results also show that DRFS does not suffer a bandwidth bottleneck problem and performs better than NFS as soon as at least 3 nodes access a file concurrently. Thus, DRFS performs at best for files larger than 1 MB. DRFS has the advantage of avoiding a SPOF (Single Point of Failure) and higher data availability.

Future work will investigate the possibility to introduce user access rights. Furthermore, the evaluation of performance by using more sophisticated read/write usage patterns and the impact of fragment size on DRFS performance is to be evaluated. Feature-wise, the support for hard links is also foreseen.

7.3 Incentives against Hidden Action in Storage Overlays

The design of delay-sensitive storage overlays over best effort networks is still an open research problem [136], [137], [138]. In this Section, we approach the problem of transaction delay as an observable indicator of peer behaviour: although the peer-experienced read and write times will depend on the underlay network conditions, they will be also affected by the behaviour of the serving overlay nodes. We seek to propose an incentives mechanism that provides an incentive for server peers to deliver their advertised levels of effort, even if client peers are unable to unequivocally determine if the unsatisfactory outcome of an I/O transaction is due to insufficient server effort or due to network variability. We will focus on peer-to-peer overlays with strategic, selfish peers.

Although there is a large amount of work on the economic and game theoretical modelling of peer-to-peer systems (in particular as it relates to the provision of incentives against the *free rider problem* [139], [140], [141], [142], [143]), it is frequently assumed that peers can determine the level of server effort by observing their own experienced QoS. Based on this, peers choose the service quality that they provide to other peers as a function of the service quality these other peers have, in turn, provided to the system.

Unfortunately, clients are very often unable to observe the actual server effort, since it is obscured by shifting network and peer conditions that affect transfer operations. The design for transaction quality contracts between peers connected through a best-effort network has been, thus, less widely studied.

The main difficulty in the analysis of peer-to-peer transaction quality contract design is that the effect of the behaviour of the server is *externalised* to the client. This means that if a server peer fails to deliver a high-enough contribution effort, and this leads to long wait times for the client, the server does not naturally bear the negative consequences of this action. Instead, these consequences are externalised to the peer who requested the service.

Since the client is unable to directly measure the actions of the server, the fact that the server might have not delivered its advertised service quality can be only indirectly inferred. In the present section we propose a *principal-agent* model for hidden action that gives server peers sufficient incentives to meet their advertised effort levels, without the peers having to decide for each transaction whether the observable outcome (the actual resolution time linked to the transaction) was due to server behaviour or network conditions. This is accomplished by allowing peers to draft contracts that provide incentives for truthful revelation of system contribution capabilities. This helps strategic peers to obtain more predictable service levels. We propose an algorithm for the calculation of the probabilistically optimal payments on a transaction outcome quality dependent contract offering greater payment if the transaction is concluded with low delays. Although we propose a general expression for the calculation of these payments, every particular application will require its own probabilistic model of the relationship between the effort that the server puts into a transaction, the prevailing network conditions and the transaction quality experienced by the client.

7.3.1 Modelling Hidden Action

The term *hidden action* is used when, in a two party transaction, the actions of one of the parties (usually regarding contract compliance or lack thereof) are hidden from the other. One model for these cases is the *principal-agent* model, where an economic actor (the *principal*) trusts another one (the *agent*) to perform a task. However, the actions that the agent takes (or fails to take) in the context of the task are not completely verifiable by the principal. Thus, the agent has the power to impose, through its actions, an externality on the principal. To transfer some of this risk back to the agent, the principal will usually require a contract by which the value that the agent obtains from the transaction will depend on the observable consequences of its actions. This contract is designed to transfer at least part of the externality back to the agent. Additionally, this contract creates an “audit point” where the principal is able to assess the actions of the agent, and respond accordingly.

In the case of peer-to-peer storage overlays, the client plays the part of the principal and the server plays the part of the agent: the client trusts the server to complete an I/O request according to its advertised effort, but the experienced transaction latency will also depend on the variability of the network on which the service takes place. Thus, the server will have some degree of plausible deniability if its service is unsatisfactory, as degradation might be indicative of fluctuating network conditions and not lack of effort on its side.

In order to use the principal-agent model, we must assume the existence of a *market and currency system* $\mathcal{M}$. The role of $\mathcal{M}$ includes informing all peers of the effort guarantees advertised by other peers, communicating pricing and resource availability information, securely maintaining the amount of currency that peers possess, and executing currency transfers between peers. Thus, $\mathcal{M}$ is the open market on which peers advertise their resources and formalise transactions. We assume that services are found, purchased and paid for using currency and procedures defined in $\mathcal{M}$.

There is a vast body of research regarding the distributed implementation of $\mathcal{M}$ using either structured or unstructured overlays ([144], [145], [146], [147], [148], [149], [150], [151], [152]) and our technique can be easily adapted to be used on any one of these systems[2]. Instead of proposing our own implementation of $\mathcal{M}$, we propose an open model applicable to a wide range of possible $\mathcal{M}$ implementations, and leave its precise modelling and analysis outside the scope of this deliverable.

Through $\mathcal{M}$, clients search for prospective servers according to the QoS characteristics that the servers themselves advertise. This takes the form of either a distributed hash table (DHT) query, or directed broadcast on an unstructured overlay. Of course, servers will only truthfully reveal their true transaction delay capabilities if the benefit they can obtain if they do so is greater that the benefit they can obtain if they lie.

A peer i with a server role will only provide a service to a client j if the utility that a transaction with j offers is at least as large as the average utility that i could obtain from a transaction with any other peer k. Thus, if we define U_r as the average utility that a peer can expect as a result of a transaction, we have that the minimum utility that i will accept from an interaction with j is U_r (we call this the *rationality* condition).We assume that peers can query $\mathcal{M}$ to learn U_r and that U_r is updated by $\mathcal{M}$ with every successful transaction.

Once i has accepted a request from j, it has to determine the treatment it will give to it: i might give different priority to different requests, different requests might pertain to different local indexes of varying sizes, or the peer might have other local processes competing for scarce CPU or upload bandwidth resources. Thus, i can respond to the request from j with various levels of *effort* $\phi \in R_{\geq 0}$. The service quality $q \in R_{\geq 0}$ that j experiences as an outcome of the transaction will be contingent on ϕ: higher values of q are positively correlated with higher values of ϕ[3].

[2] We assume that peers have reliable identities, and thus we do not address Sybil or whitewashing attacks [153], [154].

[3] We are deliberately vague in the definition of ϕ and q, as they are application-dependent. For some overlays ϕ and q might be defined in terms of delay, while for others they might be better defined in terms of jitter and average throughput.

Table 7-2. Principal-agent model notation.

ϕ	Level of effort by the server
U_S	Utility function for the server
U_C	Utility function for the client
q	QoS enjoyed by the client
ψ	Payment given by the client to the server
p	Probability of client getting high quality service as a function of the effort by the server.
U_r	Reservation utility for the server

However, we assume that the client is unable to observe ϕ - this is *private information* of the server. Additionally, we assume that there is uncertainty in transaction quality outcomes: due to varying delay and throughput conditions in the network, it is impossible for *j* to infer ϕ directly from the observed transaction quality *q*. Thus, an unobservable decision by *i* (its choice of ϕ) has an effect on the utility experienced by *j*, and we have a situation where the principal-agent model applies [155]. The client is unable to unambiguously infer the behaviour of the server from the observation of the transaction outcome, and thus is vulnerable to exploitation by it. Thus, the server needs to be given an incentive (in the form of an outcome-dependent payment $\psi \in R_{\geq 0}$ measured in the currency units of $\mathcal{M}$) to give its best effort by assimilating some of the risk associated with the network conditions that might affect the transaction outcome. These variables are summarised in Table 7-2.

7.3.2 The Principal-Agent Model

The objective of the model is to obtain the payments that the client needs to offer the server as a function of the quality of service it receives, in order to ensure that the server puts maximal effort into maintaining its advertised effort levels. The client thus calculates payments as a function of the server-advertised effort (and the estimated condition of the overlay network link between them) and creates a contract for the server. The server can either accept the contract as it stands, or reject it immediately.

If the server is able to maintain the effort commitments that it advertised in $\mathcal{M}$, it is considered to have devoted *high effort* to the transaction, and $\phi = \phi_+$. On the other hand, if the server is unable to maintain its advertised effort, it is considered to have devoted *low effort*, and $\phi = \phi_-$. Of course, low levels of effort increase the chance of the outcome having low QoS, and we seek to create an incentive mechanism against them.

A transaction is defined to have failed if the quality that the client experiences is lower than the minimum quality that the client is willing to tolerate. Thus, for any successful transaction, the outcome for the client can be either *high quality* (and $q = q_+$) or *low quality* (with $q = q_-$). The outcome of the transaction (the service quality experienced by the client) depends probabilistically upon the level of effort of the server:

$$\begin{aligned} P[q = q_+ | \phi = \phi_+] &= p_+ \\ P[q = q_- | \phi = \phi_+] &= 1 - p_+ \\ P[q = q_+ | \phi = \phi_-] &= p_- \\ P[q = q_- | \phi = \phi_-] &= 1 - p_- \end{aligned}$$

We denote with ψ the payment that the server will receive after the transaction is completed. The client will pay the server a differentiated amount, according to the quality that it experiences. Thus, the client will pay the server $\psi = \psi_+$ if the outcome is high quality, a smaller amount ψ_- if the outcome is low quality, and nothing if the transaction fails. As stated, $\phi_+ > \phi_-$, $\psi_+ > \psi_-$ and $q_+ > q_-$.

We continue the derivation of the optimal contract payments by defining the following utility function for the client:

$$U_c(q, \psi) = q^\beta - \psi$$

where $\beta \in (0,1)$. Let U_c^+ be the expected utility for the client if $\phi=\phi_+$, and U_c^- the expected utility for the client if $\phi=\phi_-$. Then, we have that:

$$\begin{aligned} U_c^+ &= p_+ U_c(q_+, \psi_+) + (1 - p_+) U_c(q_-, \psi_-) \\ U_c^- &= p_- U_c(q_+, \psi_+) + (1 - p_-) U_c(q_-, \psi_-) \end{aligned}$$

The client wishes the server to put high effort in the transaction, and thus to calculate ψ_+ and ψ_- to maximise U_c^+ (we shall not consider optimising U_c^-). We define the following utility function for the server:

$$U_s(\psi, \phi) = \psi^\alpha - \phi$$

where $\alpha \in (0,1)$. Then, the expected utility for the server as a function of its effort and payment is:

$$\begin{aligned} U_s^+ &= p_+ U_s(\psi_+, \phi_+) + (1 - p_+) U_s(\psi_-, \phi_+) \\ U_s^- &= p_- U_s(\psi_+, \phi_-) + (1 - p_-) U_s(\psi_-, \phi_-) \end{aligned}$$

We seek to ensure that the server obtains a higher utility by exerting ϕ_+ than by exerting ϕ_- (we call this the *incentive compatibility* constraint). Thus, we must find the ψ_+ and ψ_- that solve the following optimisation problem:

$$\begin{gathered} \text{Maximize:} \quad U_c^+ \\ \text{Subject to:} \quad U_s^+ \geq U_r \quad \text{(rationality)} \\ \text{And:} \quad U_s^+ \geq U_s^- \quad \text{(incentive compatibility)} \end{gathered}$$

It can be shown (see Section 7.3.3) that the optimal payments for the outcome quality-dependent contract are:

$$\psi_- = \left(U_r + \frac{p_+\phi_- - p_-\phi_+}{p_+ - p_-} \right)^{\frac{1}{\alpha}}$$

$$\psi_+ = \left(U_r + \frac{(1 - p_-)\phi_+ - (1 - p_+)\phi_-}{p_+ - p_-} \right)^{\frac{1}{\alpha}}$$

7.3.3 Optimal Contract Prices for a Hidden Action Model

The optimisation problem introduced in the previous Section is equivalent to:

$$\begin{aligned}
\text{Maximize:}\quad & p_+(q_+^\beta - \psi_+) + (1-p_+)(q_-^\beta - \psi_-)\\
\text{Subject to:}\quad & p_+\psi_+^\alpha + (1-p_+)\psi_-^\alpha - \phi_+ \geq U_r\\
\text{And:}\quad & p_+\psi_+^\alpha + (1-p_+)\psi_-^\alpha - \phi_+\\
& \geq p_-\psi_+^\alpha + (1-p_-)\psi_-^\alpha - \phi_-
\end{aligned}$$

We construct the following Lagrangian:

$$\begin{aligned}
L(\psi_+,\psi_-) = \quad & p_+(q_+^\beta - \psi_+) + (1-p_+)(q_-^\beta - \psi_-)\\
& + \lambda(p_+\psi_+^\alpha + (1-p_+)\psi_-^\alpha - \phi_+ - U_r)\\
& + \mu((p_+ - p_-)(\psi_+^\alpha - \psi_-^\alpha) - \phi_+ + \phi_-)
\end{aligned}$$

The critical point requirement $\nabla L = 0$ yields the following first-order conditions:

$$\alpha\psi_+^{(\alpha-1)}(p_+\lambda + \mu(p_+ - p_-)) = p_+$$
$$\alpha\psi_-^{(\alpha-1)}((1-p_+)\lambda + \mu(p_+ - p_-)) = (1-p_+)$$

$$p_+\psi_+ + (1-p_+)\psi_-^\alpha = U_r + \phi_+ \qquad \textit{Eq. 7-1}$$
$$(p_+ - p_-)(\psi_+^\alpha - \psi_-^\alpha) = \phi_+ - \phi_- \qquad \textit{Eq. 7-2}$$

By obtaining ψ_+^α from $p_+\psi_+ + (1-p_+)\psi_-^\alpha = U_r + \phi_+$ Eq. 7-1 and substituting on $(p_+ - p_-)(\psi_+^\alpha - \psi_-^\alpha) = \phi_+ - \phi_-$ Eq. 7-2, we find that:

$$U_r + \phi_+ - \psi_-^\alpha = \frac{p_+(\phi_+ - \phi_-)}{p_+ - p_-}$$

Thus:

$$\psi_- = \left(U_r + \frac{p_+\phi_- - p_-\phi_+}{p_+ - p_-}\right)^{\frac{1}{\alpha}} \qquad \textit{Eq. 7-3}$$

By substituting Eq. 7-3 on Eq. 7-2 we find that:

$$\psi_+ = \left(U_r + \frac{(1-p_-)\phi_+ - (1-p_+)\phi_-}{p_+ - p_-}\right)^{\frac{1}{\alpha}}$$

7.4 Defending the Sybil attack: a study of protection mechanisms in KAD

Our original design of the revocation mechanism [156] suffers of security issues based on a weakness in the identification of peers affecting many DHTs and allowing Sybil attacks. The Sybil Attack consists in creating a large number of fake peers called the ”Sybils” and placing them in a strategic way in the DHT to take control over a part of it. Our recent work tries to evaluate this issue in the KAD network.

7.4.1 KAD and the Sybil attack

KAD is a structured P2P network based on the Kademlia distributed hash table routing protocol. It is implemented by the popular open source file sharing application eMule and is one of the widest deployed structured P2P network.

However, previous studies [157] have shown that KAD can be easily and hardly damaged with a Sybil attack launched from a single computer. To prevent Sybil attacks, the authors have recently implemented new protection mechanisms that we have evaluated.

There are three main mechanisms trying do mitigate the Sybil attack. Firstly, a packet tracking mechanism avoids unwanted messages to be further considered by the application and allows detecting and discarding flooding peers. Secondly, an IP address limitation is set to avoid multiple KADIDs using the same IP address in the routing table of a peer, which is typically the case when a Sybil attack is launched. Finally, the IP address and the KADID of a peer are checked before being considered by another. This is done to avoid identity spoofing that can be used in more sophisticated attacks.

7.4.2 Evaluation of the Protection Mechanisms

We evaluated the efficiency of the protection mechanisms by designing and adapting an attack for several KAD clients with different levels of protection.

7.4.2.1 No Protection

The first client attacked was unprotected resulting to the implementation of the simplest and most effective Sybil attack possible. Sybils KADIDs are forged regarding the KADID of the targeted peer in order to best match its routing table. Sybils are announced with a rate of 4 per second. This version of the client implements an optimisation that becomes a major design failure when considering the Sybil attack as it keeps all peers with the same IP address alive with a single message. So, a single Hello_RES message is sufficient to keep hundreds of Sybils alive.

Our results below show that the routing table of the unprotected version is almost fully corrupted under attack: at the end of an attack during 300s, the routing table of the unprotected client counts 70% of Sybils (689 sybils for 953 contacts) when the protected client is not affected. The major part of the routing table that is not polluted is made up of 200 good contacts saved from the previous execution and pre-loaded to bootstrap.

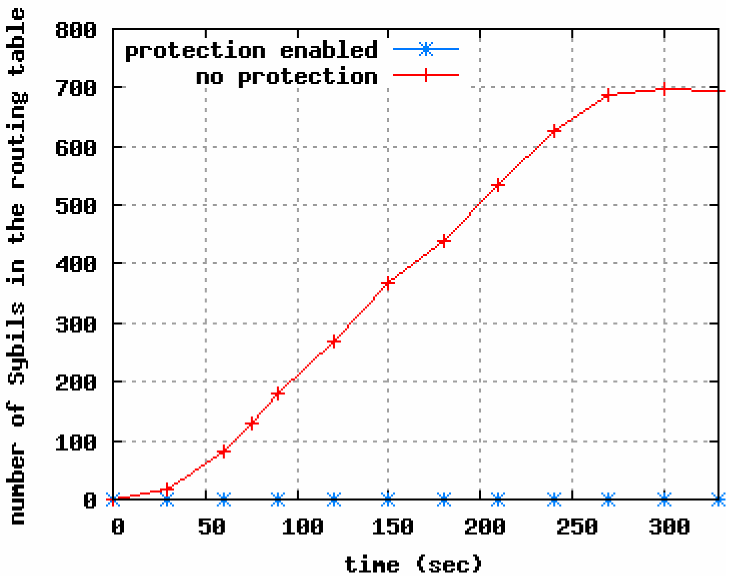


Figure 7-7. Propagation of Sybils in the routing table.

7.4.2.2 Flooding Protection

The second experiment aims to evaluate the flood protection. So, we launched an attack against a client with the IP limitation disabled. To be successful, the attack had to be improved in two ways. First, we modified our attack to keep each Sybil alive by responding to the Hello_REQ messages from the victim as the weakness described before is no more possible. The problem is that Hello_REQ messages do not include the targeted KADID whereas Hello_RES messages need it. A normal peer just has to manage its own KADID so that it clearly knows what to respond to a Hello_REQ message. On the contrary, our Sybil attack manages multiple identities and does not know which KADID is addressed by the Hello_REQ message. To bypass this limitation and find with which KADID to answer, each Sybil communicates on a different UDP port so that we can retrieve the wanted KADID. With this modified attack, we are able to maintain Sybils alive in the long run. Secondly, we sent Hello_REQ messages every 30 seconds to be under the flooding threshold.

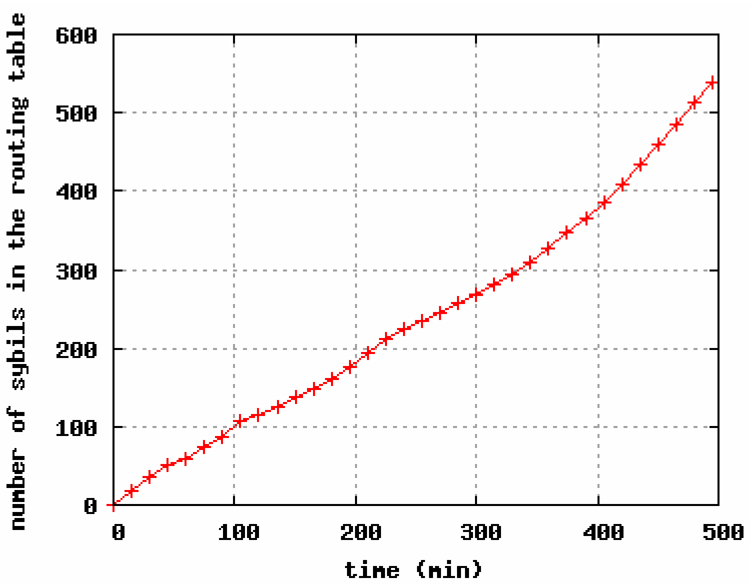


Figure 7-8. Propagation of Sybils in the routing table with flood protection enabled.

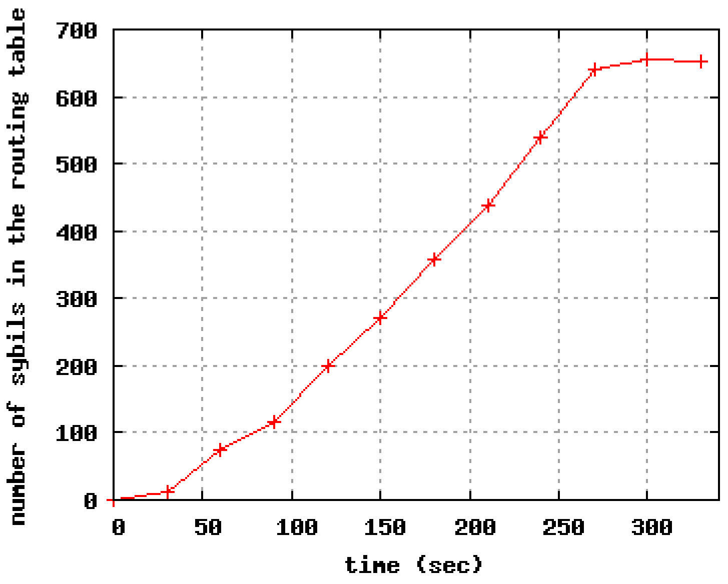


Figure 7-9. Propagation of Sybils with spoofed IP addresses with flood & IP limitation protections enabled.

Our experiments show that the flooding protection works as expected. The limitation rate before dropping Hello_REQ is set to 3 packets per minute and above 15 packets per minute, our IP address is banned. The results are shown in the figure below. We can see that even if the attack is clearly slowed down, taking more than 8 hours to inject the same number of Sybils where previous attack only took few minutes, the routing table is still polluted with more than 60% of Sybils (540/873). The flooding limitation can protect the majority of short connections to the KAD network, but longer connections still suffer from a widely infected routing table.

7.4.2.3 Adding IP Limitation

Our previous attacks were performed from a computer using a single IP address. With IP limitation enabled, we effectively measure that only the first Sybil enters the routing table, the other KADIDs presenting the same IP address can not pass. To bypass this limitation, we modified our attack using raw sockets so that each Sybil is announced with its own randomly spoofed IP address. Figure 7-9 shows that Sybils with spoofed IP addresses totally break this protection. We also noticed that the flooding protection is also completely bypassed by the IP spoofing. In fact, the packet tracking used to detect flooding measures the number of incoming requests from a given IP address. When spoofing IP addresses, each message appears to be from a different source so that the flooding is not detected.

7.4.2.4 Adding Identity Verification

The attack launched to test the identity verification mechanisms is the same as the one described above, using Sybils with spoofed IP addresses. Part of the behaviour of this mechanism seems strange because the Sybils that fail the three-way handshake are still added (at least temporary) in the routing table and marked as "unverified", taking the place of potentially good contacts in the routing table. The positive aspects are that all Sybils are marked as unverified and that unverified contacts are dropped faster from the routing table than with the classical 1-hour timeout. Moreover, unverified contacts can not update themselves before being checked, so that Sybils with spoofed IP addresses really are on limited time. Moreover, KADID overwriting is also impossible without the proper private key.

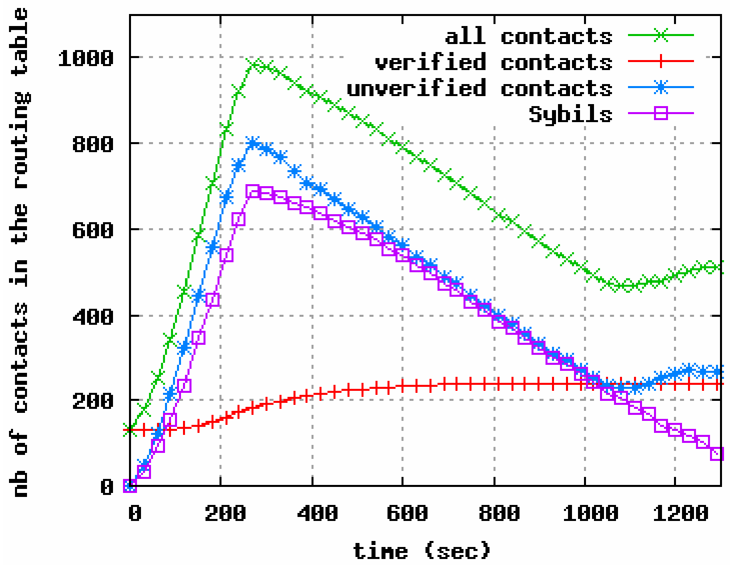


Figure 7-10. Evolution of the contacts status under Sybil attack.

Finally, the results obtained with the very last version show significant improvements of the Sybil attack defence. As expected, IP address limitation coupled with identity verification avoids the Sybil attack from a single source when it previously would have been able to infect and spy or DoS the all network.

7.4.2.5 Distributed Sybil Attack

At this point, we have tested the resilience of the routing table to Sybils with the new protection mechanisms. But a massive Sybil attack is not the only and smartest way to damage the network. More localised Sybil attacks are possible and need fewer resources. The eclipse attack is another well-known issue [157] of the KAD network. It consists in placing few malicious peers very close to the target ID in the DHT and making them known. Prior to the attack, the authors used a "crawler" in a tolerance zone to know many contacts and promote their Sybils. As all requests toward the target will pass through this zone where the peers are polluted, and considering the KAD search algorithm "return the X closest nodes...," the requests will arrive on malicious nodes with a high probability.

At the heart of the eclipse attack is the possibility to freely choose its KADID very close to a target. This issue is not yet handled in the last versions of KAD. So, the eclipse attack remains possible even if the IP limitation now requires its distribution (involving several public IP addresses). We realised a distributed eclipse attack with 24 nodes from EmanicsLab (slice inria_p2psybilattack). Our client is based on aMule 2.2.2, compiled to run as a daemon and modified in several points to make the attack. Our client uses a forged KADID to be very close (96 bits) to the hash of a given keyword. For this specific ID, our client spies the incoming requests and eclipses the keyword by answering to Publication requests and denying Search requests.

We first launched our attack on the keyword "the", known to be the most indexed keyword in the network. Despite a massive indexation of publish requests (16000 requests per minute), we cannot totally eclipse the keyword. After investigation, we found out that "the" was the target of another Sybil attack involving 256 Sybils sharing 120 bits with the keyword but not denying any Search requests. In a second time, we launched our attack on the keyword "document", receiving "only" 115 publish requests per minute but resulting in its total eclipse. Even if the needed bandwidth depends of the size of the attack and the popularity of the targeted ID, it keeps very small (few KB/s by node).

7.4.3 Conclusions

The local solution chosen by the authors of eMule fits the constraints of the network: no infrastructure cost and a backward compatibility. Our results show that the protection really makes the Sybil attack harder to realise and limits its impact. Where an attack could have been launched from a single computer few months ago, it now needs a botnet or a far more complex distributed architecture to achieve the same results.

However, several limitations still remain as the eclipse attack does not require substantial resources and keeps being a real threat as long as the KADID can be freely chosen and the malicious nodes placed very close to the target.

Even if not perfect, the defences against the Sybil attack studied in this deliverable are absolutely generic and can be easily applied to every structured P2P network. We think that this kind of "common-sense" protection is the minimum that every implementation of a structured P2P network should have, unless being totally unaware of the Sybil attack issues.

This work has been submitted in [158]. Before coming back to revocation, our future work will consist in designing new constraints improving the protection to avoid any attack taking control over a part of the DHT.

7.5 Crawling Bug Tracker for Semantic Bug Search

Trouble ticket systems and bug tracking systems are widely deployed in the information technology industry. Software and hardware companies use bug tracking systems during the development cycle to track bugs and design issues, or during later phases of the product life-cycle to keep track of defect reports and to obtain quality indicators. Almost all large open source projects maintain online bug tracking systems. In addition, there are many bug tracking systems supporting people who package open source software components for various software distributions. Some companies provide special online support forums (also known as communities or knowledge bases) for their products that often resemble bug tracking systems.

The fast growing amount of online information that can be used to resolve problems has led to a situation where system administrators and network operators often use search engines in order to find hints how to resolve a specific problem. However, it is our experience that searching in this way is not as efficient as we would like it to be because generic search engines do not seem to take advantage of the data found in trouble ticket or bug tracking systems. On the contrary, a domain-specific search engine can dive into many bug trackers simultaneously and thus save the user a lot of manual effort. Second, by having a view of the combined data of several trackers, a domain-specific search engine can discover and utilise relationships and similarities between bug reports that exist in different bug trackers.

Trouble ticket systems and bug tracking systems contain semi-structured data. Predefined fields are used to keep track of the status and meta-data associated with a problem report while textual descriptions are used to describe the problem and to document the process for resolving the problem. This data can be used to construct specialised search systems that have access to and knowledge of meta-data out of reach to general search engines. It can also be used to build automated reasoning systems that help to diagnose problems based on past experience. This report presents our work towards a distributed search engine for bug reports, namely Buglook. The aim of Buglook is to find relevant information quickly and to cope with the fact that the relevance of bug records changes quickly, due the short lifecycles of today's software products and services. The work in this report has been published in [159], [160] and [161].

7.5.1 Bug Tracking Systems

Trouble ticket systems (TTS) have been widely used by network operators in order to assure the quality of communication services. A bug tracking system (BTS) is a special trouble ticket system used to keep track of software bugs. BTSs in general aim to improve the quality of software products. They do so by keeping track of current problems, and maintaining historical records of previously experienced issues. They also establish an expert system that allows searching for similar past problems, and providing reports and statistics for performance evaluation of the services [162].

We explore the features of several BTSs, focusing on several properties that are important for obtaining data from them. For each BTS we checked whether the BTS supports the functionality in question, and if so, which bug sites (i.e., specific installations of a BTS) support that function. A small sample of our results for BTSs and sites is given in Table 7-3 and Table 7-4 respectively. An explanation of each column in these tables is given below.

Table 7-3. Overview of bug tracking systems and some of their features (as of September 2008). Items marked with * are optional for each site. # reads dependencies.

Tracker	License	Access	Updates	Schema	Dept. #	Search
Bugzilla	MPL	HTML, XML-RPC*	SMTP, RSS*	Textual	Optional	Filter, keywords
Mantis	GPL	HTML, SOAP*	SMTP, RSS	Graphical	Yes	Filter
Trac	BSD	HTML	SMTP, RSS*	Graphical	No	Filter, keywords
Debian BTS	GPL	HTML, SMTP	SMTP	Unknown	Optional	Filter
phpBugTracker	GPL	HTML	SMTP	Textual	Yes	Filter, keywords
Flyspray	LGPL	HTML	SMTP,RSS, XMPP	Unknown	Yes	Filter, keywords

Table 7-4. Some popular bug tracking sites (as of September 2008). + indicates that we were unable to get precise numbers and our numbers present a lower bound. # reads dependencies.

Site	System	Version	Bugs	Activity	Custom	RPC	RSS	Dep. #
Bugs.debian.org	Debian	N/A	349346	1036	N/A	No	No	No
Bugs.kde.org	Bugzilla	Unknown	166403	1046	Light	No	No	No
Bugs.eclipse.org	Bugzilla	Unknown	248295	2825	Heavy	Yes	Yes	Yes
Bugs.gentoo.org	Bugzilla	Unknown	211036	1268	None	No	Yes	Yes
Bugzilla.mozilla.org	Bugzilla	3.2rc1+	448106	4052	None	Yes	Yes	Yes
Bugzilla.redhat.com	Bugzilla	3.1.4+	215051	2850	Light	Yes	Yes	Yes
Qa.netbeans.org	Bugzilla	Unknown	146344	2243	Heavy	No	No	Yes
Bugs.digium.com	Mantis	Unknown	13416	59	None	No	Yes	Yes
Bugs.scribus.net	Mantis	1.0.7	7148	34	None	No	Yes	Yes
Bugtrack.alsa-project.org	Mantis	1.0.6	4120	15	None	No	No	Yes
Dev.rubyonrails.org	Trac	0.10.5dev	11446	Unknown	None	No	Yes	No
Trac.edgewall.org	Trac	Unknown	7326	Unknown	None	No	Yes	No
Bugs.icu-project.org	Trac	0.10.4	4324	Unknown	None	No	Yes	No

For each BTS, we looked at how the system is licensed for use, how its collection of bugs can be accessed, whether it is possible to easily receive notification of updated data, whether the database schema used by the BTS was available, whether the BTS keeps track of bug dependencies, and whether the BTS can be searched for bugs with a given property. The most important function of a BTS is retrieving bug reports in their most current states. All systems must at a minimum have an HTML-based web interface, but convenient automated retrieval requires a formalised programmatic interface, either based on XML-RPC or SOAP. While all BTSs support e-mail (via SMTP) as an update notification mechanism, e-mail is non-trivial to use for a web application. Some systems support RSS or Atom feeds, which are significantly easier to use by a program.

In order to understand the structure of the information stored in a BTS, we investigated whether the underlying data model is documented. Some systems provide this information in a textual format while others provide graphical representations, usually in ad-hoc notations. Some systems do not provide a clear description of the data model underlying the BTS and it is necessary to reverse engineer the data model by looking at concrete bug reports.

Based on popularity and available documentation, we chose to focus on Bugzilla, Mantis, Trac and Debian BTS. With the exception of the Debian BTS, which is only used for the Debian operating system, the BTSs we chose all publish lists of known public sites that use them for bug tracking. Starting from these lists, we investigated all sites that are accessible and did not require authentication to browse their repositories (about 85 sites). Table 7-4 lists the sites with the largest number of bugs for each BTS. For each site, Table 7-4 shows which version of what BTS is used, how many bugs are stored there in total and how many have been added in one week, indicating the activity of the site. The table also specifies whether the site has been customised from its base BTS, whether it supports a programmatic XML-RPC interface or RSS feeds, and whether the site supports bug dependency relations.

The version of the underlying BTS largely impacts the set of available features. For example, Bugzilla only supports RSS feeds as of version 2.20, and XML-RPC as of version 3.0. The number of stored bugs and the rate of opening new bugs indicate the popularity and activity of a site. The margin between the most popular Bugzilla sites and the most popular sites using other BTSs is very large. We believe that the reason is that Bugzilla was the first widely-known open-source BTS when it was released in 1998. Mantis was only started in late 2000, and Trac is even newer.

Some sites customise their BTS in order to provide better integration of the bug tracker into the rest of their web site. While some sites only change the visual appearance of the BTS, others also modify the functionality of the BTS. Customised sites pose a problem for automated bug retrieval: a system that is designed to derive structured data from presentational HTML will generally fail to handle a significant change of a site's appearance.

Programmatic interfaces in protocols like XML-RPC or SOAP can be used by programs to query a bug tracker for structured data, without having to guess the value of any fields presented in human-readable form (HTML, SMTP). While this greatly simplifies interfacing to that bug tracker, no BTS currently makes such an interface a default option. It is an optional feature of the BTS at best, and not supported at all at worst.

7.5.2 Unified Data Model

In order to integrate bug data from different BTSs into a single bug database, we define a unified data model for bugs. This model must be simple and easy to use for our purpose; it is not necessary to be able to represent all available details of all systems. Our investigation of the database schemas of the four BTSs we considered exposes several interesting observations. The bug formats of Bugzilla, Trac and Mantis share many similar fields that can be classified in two main groups:

- The administrative meta-data associated with a bug is often represented as field-value pairs with very precise semantics, such as *id*, *severity*, *reporter*, *summary*, among others.
- The descriptions detailing the bug and any follow-up discussion or actions are typically represented as free-form (i.e., non-formal) textual attachments.

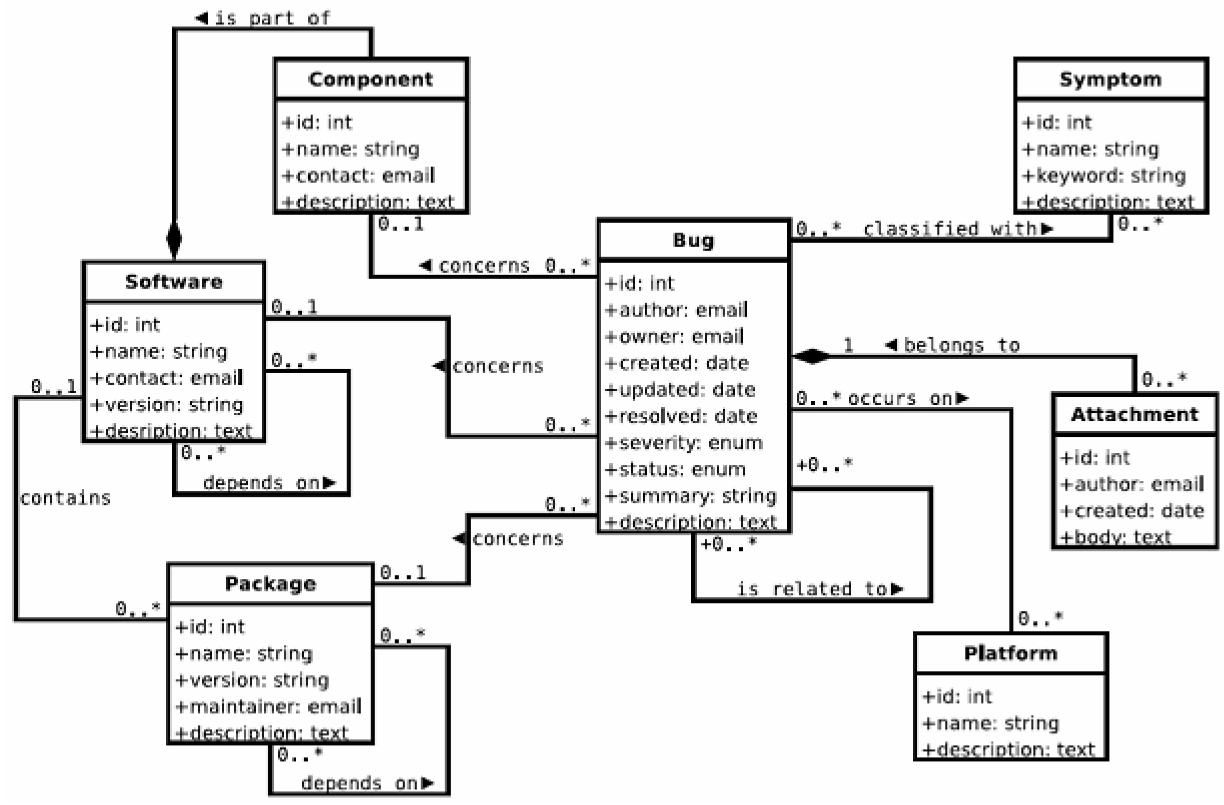


Figure 7-11. Unified bug data model represented as a UML diagram.

Table 7-5. Severity of bugs.

Unified Model	Bugzilla	Trac	Mantis	Debian
Critical	Blocker, critical	Blocker, critical	Block, crash	Critical, grave, serious
Normal	Major	Major	Major	Important, normal
Minor	Minor, trivial	Minor, trivial	Minor, trivial, text, tweak	Minor
Feature	Enhancement	N/A	Feature	wishlist

Because the unified data model is used to support semantic search, we aim to extract fields from bug reports in such a way as to minimise the loss of bug information. We introduce new fields that establish the relationships between bugs or provide for more sophisticated classification. The values of these fields can be derived differently for each BTS: Mantis users can manually specify the relation of a bug to other bugs when reporting it; Debian users can indicate which package a bug relates to; and Bugzilla and Trac offer a keyword field that enables the classification of bugs. To exploit data from a bug's description and its attachments, we use several text processing techniques [163], [164].

Figure 7-11 shows our unified bug data model in the form of a UML class diagram. The central class is the *Bug* class. The *id* attribute of a *Bug* instance uniquely identifies a bug. We use the URL where a bug can be retrieved as its identifier. Most of the attributes of a *Bug* instance can be easily extracted from the retrieved data. Our *severity* attribute is probably the most interesting to fill correctly, because BTSs have very different severity classifications for bugs. Table 7-5 shows how we map the severity values of the BTSs into our data model, which only distinguishes the severity values *critical*, *normal*, *minor*, and *feature*. The *status* attribute of a *Bug* instance only has two values: the value *open* represents what BTSs call *unconfirmed*, *new*, *assigned*, *reopened* bugs while the value *fixed* represents what BTSs call *resolved*, *verified*, and *closed* bugs.

Free-form textual descriptions are modelled as *Attachment* objects. Every *Attachment* belongs to exactly one *Bug* object. Some BTSs provide information about the platforms affected by a bug. We represent platforms (such as “Windows2000" or “MacOS X") as *Platform* objects. The *Keyword* class represents keywords used to describe and classify bugs.

The left part of Figure 7-11 models what piece of software a bug is concerned with. While some BTSs are only concerned with bugs in a specific piece of software, software in larger projects is split into components and bugs can be related to specific components. The classes *Software* and *Component* model this structure. The Debian BTS is somewhat different from the other BTSs as it is primarily used to track issues related to software “packages", that is software components packaged for end user deployment. Since there is a large amount of meta-information available for Debian software packages (dependency, maintainer and version information), we have introduced a separate *Package* class to represent packaged software.

With the bug dataset, we have used meta-data search [165] for structured, domain-specific data, and semantic search [163] for textual data.

7.5.3 Full-text Search

The LSI method transforms the essential abstract concepts of a document or query into a semantic vector. To generate this vector, a document or a query is first represented in a term vector. The Vector Space Model (VSM) [164] weights each term, namely *tf-idf* weight, in the term vector by calculating the appearance frequency of this term in the document and the appearance frequency of this term in other documents, as follows:

$$\text{tf}-\text{idf} = \frac{n_{td}}{N_d}\log\left(\frac{N}{n_{dt}}\right) \qquad \textit{Eq. 7-4}$$

The term n_{td} is number of appearances of term *t* in document *d*; N_d is number of terms in document *d*; *N* is number of documents; n_{dt} is number of documents containing the term *t*. A high frequency of a term indicates the significance of the term in the document, but its significance is compensated if the term also appears in many other documents. LSI deals with noise and synonyms in a document by transforming a term vector into a semantic vector. To carry out the transformation, LSI represents all documents and terms in the documents in a $t \times d$ matrix A, where each element a_{ij} computed by Eq. 7-4 denotes the significance of term i in document j. Using singular value decomposition (SVD) [164], A is decomposed into the product of three matrices: $A = USV^T$, where $S = diag(\sigma_1, ..., \sigma_r)$ is an $r \times r$ diagonal matrix, r is the rank of A and σ_i is the singular value of A with $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r$. LSI eliminates noise and synonyms by picking up the s largest singular values resulting in reducing the rank of A; e.g., $A_s = U_s S_s V_s^T$. The optimal value of s is chosen depending on r; e.g., between 50 and 350. Semantic vectors of documents in A are indexed by the rows of V_s. Semantic vectors of new documents or queries are computed by using U_s, S_s [164]. The similarity between two vectors is measured by the cosine of the angle between these vectors. Formally, given two vectors $q = (q_1, q_2, ..., q_s)$ and $c = (c_1, c_2, ..., c_s)$ normalised with $||q|| = 1$ and $||c|| = 1$, the similarity between q and c is computed by the following equation:

$$\cos(q,c) = \sum q_i c_i \qquad \textit{Eq. 7-5}$$

7.5.4 Meta-data Search

This search method measures field-value vectors to obtain the similarity of bug reports. This vector contains several pairs of pre-defined fields and symbolic values that can be compared by several similarity methods. The global similarity method [166] focuses on the significance of field-value pairs in vectors:

$$\text{sim}(q,c) = \sum w_i \text{sim}(q_i, c_i) \qquad \textit{Eq. 7-6}$$

n is the number of features; q_i and c_i are features of cases q and c respectively; $sim(q_i, c_i)$ is the similarity between q_i and c_i, which are expressed in binary, numeric or symbolic values; w_i is a weight of the i^{th} feature such that $\sum w_i = 1, w_i \in [0,1] \forall i$. A weight is a user-defined value which exhibits the significance of a certain feature.

7.5.5 Buglook Search Engine

Buglook consists of five components. Four of them act as services that are started once and perform their work indefinitely (or until they fail). They communicate between each other when necessary. These four services, called *daemons*, are:

- The *spider daemon* collects bug reports from various BTSs and converts them into the Unified Data Model (Section 7.5.2). Each successfully converted bug report is immediately sent to the storage daemon where it is saved persistently.
- The *storage daemon* receives bug reports from the spider daemon and stores them persistently. When contacted by the math daemon, it uses its storage pool to build a document-term matrix of all bugs. This matrix, together with several other important pieces of data, is then sent to the math daemon.
- The *math daemon* is the heart of the LSI method. It performs singular value decomposition (SVD) on a term-document matrix to obtain three matrices that can be used to answer queries.
- The *query daemon* answers search queries. Once it receives the math daemon's output, it answers queries based on that dataset until it is contacted again and given updated matrices to work with.

The fifth component is a client library, called the *query client*, which submits queries to and receives results from the query daemon. Currently it is used within a web application, but it could also be used to build desktop search interfaces or be embedded into other programs.

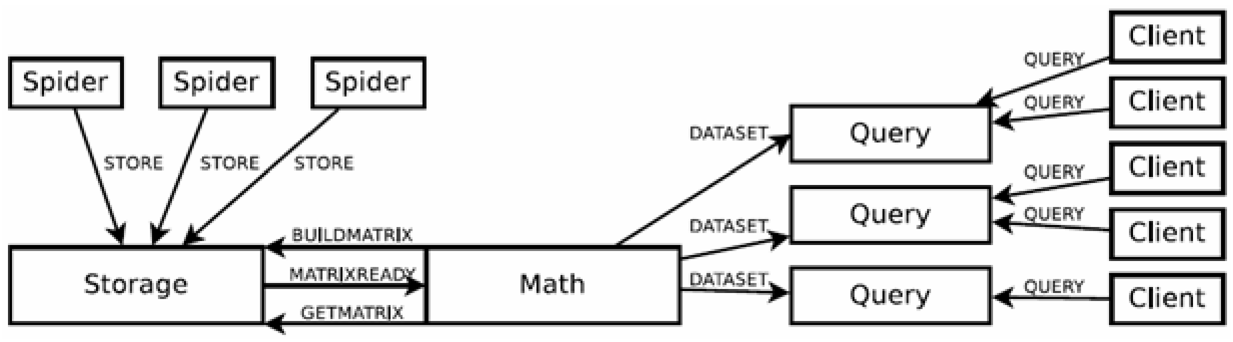


Figure 7-12. Buglook architecture: components and messages.

All Buglook components are stateless by design. Apart from a very small number of configuration settings, they do not maintain any persistent state. This is a step towards Buglook's reliability goal - if a daemon terminates for any reason, it can be restarted and it will integrate into the otherwise running system automatically. To this end, all operations that depend on other daemons are carefully watched for failure, and attempted again if they fail or time out. The overall Buglook architecture is depicted on Figure 7-12. Note that any number of spider daemons can feed the same storage system, and the math daemon's output can be used by any number of query daemons that can answer queries simultaneously as a load balancing mechanism.

The majority of Buglook was implemented in the Python programming language [167], whose strengths are portability, fast development, flexibility for easy refactoring (thanks to a weak type system), and an impressively broad standard library. Only performance-critical sections are written in C, because it allows high performance with precise control over memory management and multi-threading, at the cost of programming complexity.

All C portions of the code-base are enclosed into a single Python module called the *Buglook C eXtention (BCX)*. The storage, math, and query daemons make use of the BCX. Buglook's web interface is based on Django [168], a Python web application framework. Django is relatively new and its application programming interface is not stable at this time, but is very simple to use and integrates well into existing web server setups. Buglook additionally uses shove [169] for storage, Twisted Matrix [170] for networking and thread management, and SVDLIBC [171] for the matrix decomposition required by LSI.

7.5.6 Buglook as a Distributed Search Engine

A single instance of Buglook cannot be expected to handle downloading, synchronisation, parsing, and vectorisation of several BTSs with millions of bugs, while simultaneously maintaining an up-to-date index and performing SVD calculation of a very large document-term matrix. There are several resources that are likely to become insufficient - bandwidth, processing power, memory, and storage space. Bandwidth is needed for the initial systematic downloading of BTS sites, for periodic polling synchronisation, and for interaction with users. CPU power and memory are chiefly consumed by the singular value decomposition of the large document-term matrix. Storage space is required to cache processed bug reports and feature/semantic vectors, so that indexes can be updated.

In addition to resource deficiency, a centralised system is a single point of failure: if the hardware running Buglook fails, so will the search engine. A distributed architecture for Buglook, on the other hand, would address both performance and reliability. The question that must be answered, then, is what processes must be distributed. In this section, we outline two approaches to decentralising the search engine, which lead to several possible architectures.

7.5.6.1 Peer-to-Peer Query Propagation

One way to distribute Buglook is to keep most of its centralised structure intact, but add transparent query propagation to each instance. The resulting architecture is shown on Figure 7-13. In this scenario, a Buglook instance B_1 receives a request from a user. It performs a search procedure to answer the query locally; this is the same procedure that a centralised Buglook would execute. In parallel, B_1 forwards the same query to other instances that it knows about. These instances, recognising that the request comes from B_1 as opposed to a user, attempt to answer the query but do not forward it further. Upon receiving answers from several other instances, B_1 takes the most relevant bug reports from each list, including his own, and returns the resulting best list to the user.

The advantage of this technique is its simplicity. It can easily be implemented on top of the rest of the search engine architecture, with no consideration to the inner workings of how a query is answered. Then, given a large dataset of bugs on one or more bug sites, different Buglook instances can be configured to only consider distinct small subsets of the data. A query to one instance would still generate answers from the entire set.

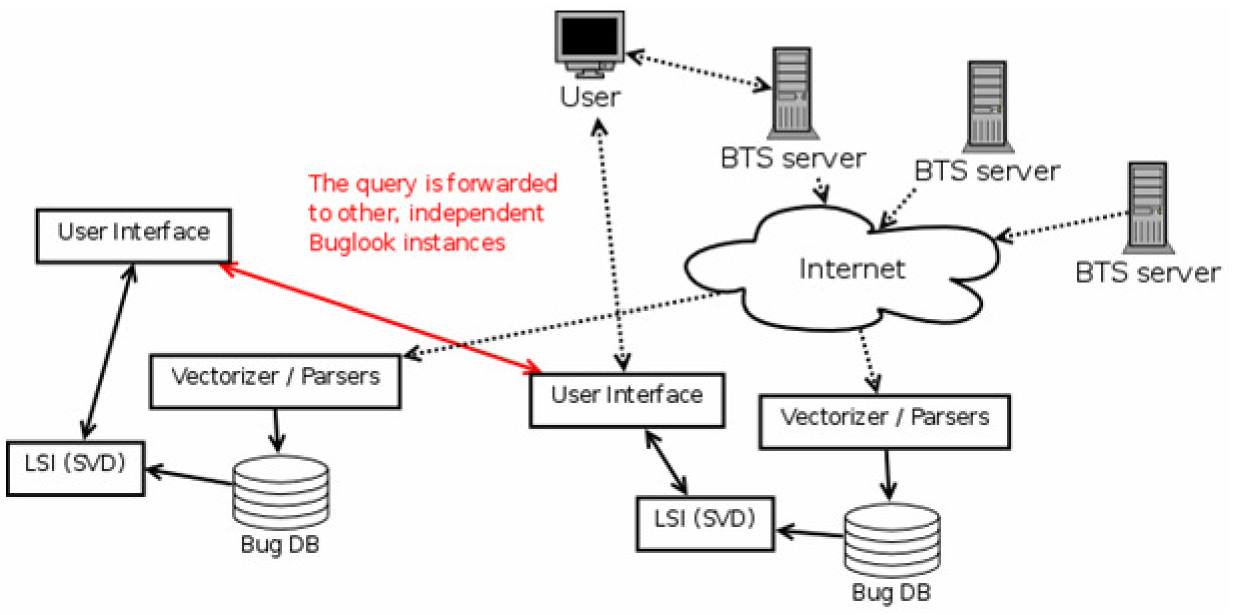


Figure 7-13. Peer-to-peer query propagation.

There are three fundamental problems with this approach. First, most information retrieval techniques, and LSI in particular, need knowledge of the entire document set to operate correctly. For instance, consider a document d_1 owned by B_1 that contains some concept c_1. Let the user ask a query that contains concept c_2. By itself, B_1 might not be able to discover that c_1 and c_2 are the same concept expressed in different words, only because it does not also own a document d_2 that contains both concepts. In this very simplistic example, a different instance that owns d_2 will return that second document, but B_1 will not return d_1.

A related problem is that even if all relevant documents could be retrieved, the Buglook instance that must compile the list of best results will not be able to easily decide which documents are best. Similarity is a relative measure, and although all documents will contain a similarity measure to a common reference (the query), that measure will have been obtained by different instances using different data, and will therefore not be directly comparable.

Last but not least, this architecture is slow. Unless there is a very high-speed network between Buglook instances, the delay introduced by query forwarding may be unacceptable. If a modern keyword search engine can search the entire web in milliseconds, then Buglook should be able to return results immediately as well.

7.5.6.2 Distributed Data Acquisition

All other architectures described below take a different approach. They answer every query locally without contacting other instances at the time of query processing. Rather, they continuously exchange data in the background to distribute their load.

The second distributed approach (Figure 7-14) is to maintain all functionality related to information retrieval (LSI, SVD, similarity calculation) independent in each Buglook instance, but to fully distribute the data acquisition step. Simply put, whenever an instance obtains some set of bug reports, it shares them with other instances, which will then not download those particular bugs.

Under ideal conditions, there is a high-speed network between Buglook instances. Such a scenario could be, for instance, a single organisation deploying a large set of Buglook nodes for load balancing and reliability, rather than single lonely instances deployed by different people in different places. Given such a high-speed network, substantial bandwidth savings can be achieved because the (slower) external link will only be used once per bug report.

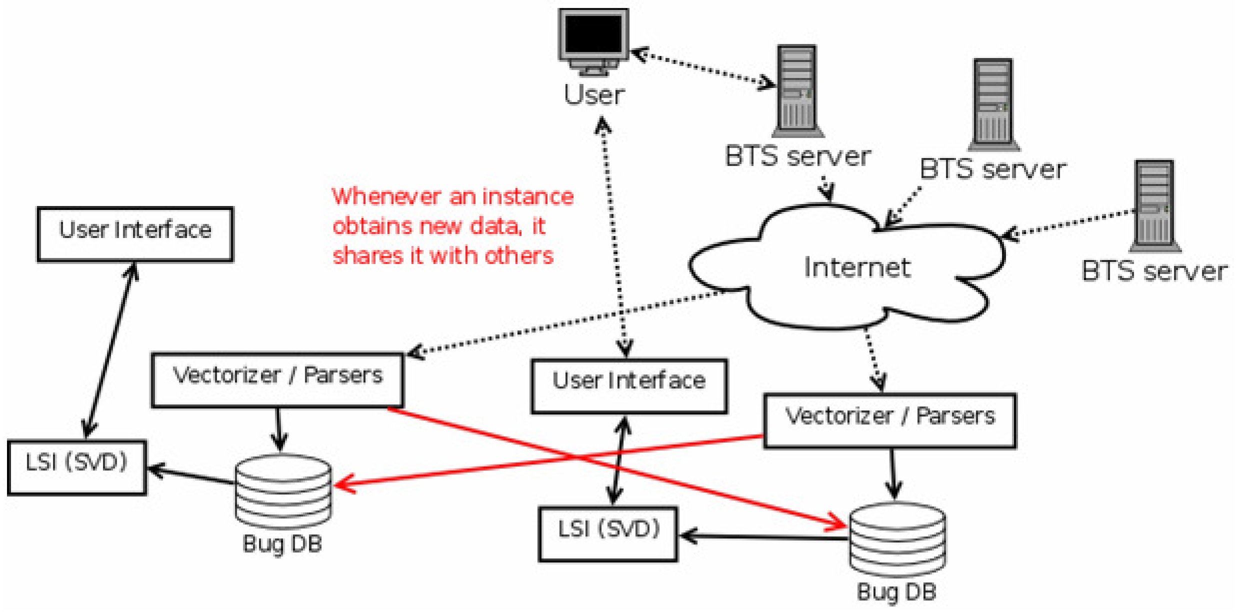


Figure 7-14. Distributed data acquisition.

Even when instances are widely distributed, this scheme has two desirable properties. First, it is friendly to the BTS sites, because a BTS will only see a single instance visiting the site and downloading bug data from it. Second, it ensures, within some delay, that all instances have the same data. This property is a prerequisite for a distributed SVD calculation. Finally, note that even over slow links, performance can be improved by compression techniques and bulk transfers of many bugs. Neither is possible when there is a BTS on the other side of the link.

The principal problem with the data sharing approach is that it does not go far enough. Every instance still stores all data locally and performs its own computations; only the network bandwidth bottleneck is eliminated.

7.5.7 Conclusions

We have provided a study of existing BTSs with a specific focus on the four most popular open source systems, namely Bugzilla, Trac, Mantis and the Debian bug tracking system. Widely used public BTSs based on these software systems contain a large number of bug reports that can be used for building bug datasets. Such datasets are invaluable for evaluating systems such as case-based reasoning engines or semantic search engines. In order to store the data in an effective way, we have developed a unified bug data model that is able to capture the most important aspects of the data maintained by the various BTSs we have analysed. Our model enables interoperable aggregation of data from different sources, useful for various purposes ranging from efficient wide-scale search to automated reasoning systems.

The multi-vector representation method [159] has been used to perform semantic search experiments on the unified bug dataset. This method exploits semi-structured bug data to search for similar bugs with salient features. The experimental results indicate that (i) the combination of full-text search and meta-data search outperforms the other combinations of full-text search and baseline search or of meta-data search and baseline search, (ii) baseline search provides less consistent and reliable results, and (iii) the bug dataset is wide and diverse in scope.

The prototype of the Buglook search system has been implemented. This system can accept a larger number of user queries for the evaluation purpose. This system also allows us to evaluate search latency and bug synchronisation on a large bug dataset. In our future work the refined and unified datasets can be used to evaluate the problem-solving capability of our distributed case-based reasoning system. Such systems will certainly be a practical tool for anyone who needs to troubleshoot a software system with a public bug tracking system.

8 Conclusions

This document provides an in-depth investigation of emerging paradigms and technologies that support the vision of autonomic management. Also, the presented research work addresses different key areas of the autonomic management field and provides state of the art approaches and technologies that exhibit several of the self-* properties explained in deliverable D9.1, which are essential for the implementation of truly autonomic management systems. Additionally, the research effort described in this deliverable builds on research work carried out in first period of EMANICS Phase 2 (presented in deliverable D9.3) and, at the same time, prepares the road for the next working period in the EMANICS project.

The excellent work presented throughout this document proves that Work Package 9 (WP9: Autonomic Management) has succeeded in establishing strong collaboration and cooperation links among WP9 partners. This also guarantees that the future outcome of our ongoing research will continue to maintain the most elevated standards and will take full advantage of the results obtained in this deliverable. As a result, we expect to see most of the activities presented in this deliverable to report a new phase in deliverable D9.5.

Policy based techniques, in particular policy conflict analysis, are intimately linked to the principles of autonomic management. Our work demonstrates that policy-based management can effectively address the issues of self-configuration and self-optimisation as two of the essential self-* properties of autonomic systems. The evaluation of our selected case-studies – QoS management and firewall configuration – shows that our proposed approaches can be effectively applied to different areas of autonomic management.

Our research work on autonomic frameworks to assist administrators in dealing with intricate fault recovery management has demonstrated that approaches based on emerging technologies such as case-based reasoning, peer-to-peer methodologies and artificial intelligence based automated planning can effectively reduce or eliminate the disruption time when a network or service fault occurs.

The complex task of brokering context information between context-aware services (CAS), which request context information and context information services (CIS) has been addressed by our proposed CoCo (Composing Context) infrastructure. Furthermore, it has also been demonstrated that our CoCo infrastructure relieves administrators from the burden posed by the management of context information since CoCo takes care of the discovery of context information services, inter-domain data transformation, and the derivation of high-level context information based on low-level context information.

We have presented an ontology-based, policy-driven architecture for managing services using the context information that provides support for customisation, definition, deployment and management. Our efforts have been directed toward providing a novel, shared ontology for the integration of contextual information into service management operations and toward supporting a policy-driven, ontology-based architecture. The Onto-CONTEXT framework demonstrates how the use of an ontology-based information model can be used for managing context information and supporting pervasive services, thus improving cross-layer management and interoperability. Additionally the OSM ontology has been proposed that defines an extensible ontology for the robust support and management of pervasive services; our efforts in this area are focused on functional middleware technologies for management.

Distributed and Autonomic Real-Time Traffic Analysis has been tackled through two innovative technologies: distributed packet capture (DiCAP) and distributed storage of IP flow records (DIPStorage). DiCAP allows for the capture of IP packet headers at high packet rates without requiring any dedicated hardware, which makes it a cost-effective solution. DIPStorage complements DiCAP by helping network operators to build a low-cost scalable storage facility that allows for the efficient analysis of IP flow records thanks to its improved performance to resolve queries on the stored records. Additionally, prototypes of both technologies have been implemented, tested and evaluated, and the obtained results prove their feasibility.

We have also presented an algorithm that allows for the provisioning of future internet services that exhibit strong emphasis on service provider collaborations. Our algorithm can potentially be used to set up electronic marketplaces for service providers. However, in order for this to happen, the systems involved in service negotiations must have management systems in charge of the negotiation process. In addition, we have demonstrated how our algorithm provides support for the service marketplace coalition formation process in such environments and have evaluated its scalability and the conditions under which the algorithm converges to optimal solutions.

Finally, we have proposed a set of various complementary solutions addressing several unsolved problems in emerging peer-to-peer storage networks. Among the addressed problems are: i) assurance of satisfactory quality of service, ii) confidence on the high performance of bandwidth requirements, iii) tracking of content changes, and iv) handling of malicious peers attempting to degrade performance and reliability.

9 References

[1] P. Flegkas, P. Trimintzios, G. Pavlou, “A policy-based quality of service management architecture for IP DiffServ networks,” IEEE Network Magazine Special Issue on Policy Based Networking, Vol. 16, No. 2, pp. 50-56, 2002.

[2] E. Mykoniati, C. Charalampous, P. Georgatsos, T. Damilatis, D. Goderis, P. Timintzios, G. Pavlou, D. Griffin, “Admission control for providing QoS in IP DiffServ networks: the TEQUILA approach,” IEEE Communications Magazine, Vol. 41, No. 1, 2003.

[3] P. Trimintzios, T. Baugé, G. Pavlou, P. Flekas, R. Egan, “Quality of service provisioning through traffic engineering with applicability to IP-based production networks,” Computer Communications, special issue on Performance Evaluation of IP Networks and Services, Vol. 26, No. 8, pp. 845-860, Elsevier Science Publishers, May 2003.

[4] M. Charalambides, P. Flekas, G. Pavlou, A. K. Bandara, E. C. Lupu, A. Russo, N. Dulay, M. Sloman, J. Rubio-Loyola, “Policy conflict analysis for quality of service management,” proceedings of 6th IEEE Workshop on Policies for Networks and Distributed Systems, Stockholm, Sweden, 2005.

[5] M. Charalambides, P. Flekas, G. Pavlou, J. Rubio-Loyola, A. K. Bandara, E. C. Lupu, A. Russo, M. Sloman, N. Dulay, “Dynamic policy analysis and conflict resolution for DiffServ Quality of Service Management,” proceedings of IEEE/IFIP Network Operations and Management Symposium, Vancouver, Canada, 2006.

[6] A. K. Bandara, E. C. Lupu, A. Russo, N. Dulay, M. Sloman, P. Flekas, M. Charalambides, G. Pavlou, “Policy refinement for IP differentiated services quality of service management,” IEEE Transactions on Network and Service Management, Vol. 3, No. 2, 2006.

[7] J. Loyola, J. Serrat, M. Charalambides, P. Flegkas, G. Pavlou, “A methodological approach toward the refinement problem in policy-based management systems,” IEEE Communications Magazine, Topics in Network and Service Management, Vol. 44, No. 10, 2006.

[8] E.C. Lupu, M.S. Sloman, “Conflicts in policy-based distributed systems management,” IEEE Transactions on Software Engineering - Special Issue on Inconsistency Management, vol. 25, pp. 852-869, 1999.

[9] D. Agrawal, J. Giles, K.W. Lee, J. Lobo, “Policy ratification,” proceedings of 6th IEEE Workshop on Policies for Networks and Distributed Systems, Stockholm, Sweden, 2005.

[10] Dunlop, J. Indulska, K. Raymond, “Methods for conflict resolution in policy-based management systems,” proceedings of 7th International Conference on Enterprise Distributed Object Computing, Brisbane, Australia, 2003.

[11] G. Russello, C. Dong, N. Dulay, “Authorization and conflict resolution for hierarchical domains,” proceedings of 8th IEEE Workshop on Policies for Networks and Distributed Systems, Bologna, Italy, June 2007.

[12] A. D. Rubin, D. Geer, M. J. Ranum, “Web Security Sourcebook,” New York, NY, USA, John Wiley & Sons, Inc., 1997.

[13] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, “Conflict classification and analysis of distributed firewall policies,” IEEE Journal on Selected Areas in Communications, Vol. 23, No. 10, pp. 2069-2084, October 2005.

[14] A. C. Kakas, P. Mancarella, and P. M. Dung, “The acceptability semantics for logic programs,” proceedings of 11th International Conference on Logic Programming, pp. 504-519, 1994.

[15] Y. Dimopoulos and A. C. Kakas, “Logic programming without negation as failure,” proceedings of the International Logic Programming Symposium, pp. 369-383, 1995.

[16] E. S. Al-Shaer and H. H. Hamed, “Firewall policy advisor for anomaly discovery and rule editing,” proceedings of 8th IFIP/IEEE Symposium on Integrated Network Management, Colorado Springs, USA, ser. IFIP Conference Proceedings, G. S. Goldszmidt and J. Schönwälder, Eds., Vol. 246. Kluwer, pp. 17-30, 2003.

[17] H. Hamed and E. Al-Shaer, “Dynamic rule-ordering optimization for high-speed firewall filtering,” in ASIACCS, F.-C. Lin, D.-T. Lee, B.-S. Lin, S. Shieh, and S. Jajodia, Eds. ACM, pp. 332-342, 2006.

[18] A. K. Bandara, A. C. Kakas, E. C. Lupu, and A. Russo, “Using argumentation logic for firewall policy specification and analysis,” in DSOM, ser. Lecture Notes in Computer Science, R. State, S. van der Meer, D. O’Sullivan, and T. Pfeifer, Eds., vol. 4269. Springer, pp. 185-196, 2006.

[19] A. Aamodt, E. Plaza, “Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches,” AI Communications, vol. 7, No. 1, pp. 39–59, 1994.

[20] Gnutella protocol specification v0.4: http://www9.limeware.com/developer/, August 2007.

[21] A. Crespo, H. Garcia-Molina, “Routing Indices For Peer-to-Peer Systems,” proceedings of 22nd International Conference on Distributed Computing Systems, Washington DC, USA, IEEE Computer Society, 2002.

[22] V. Kalogeraki, D. Gunopulos, D. Zeinalipour-Yazti, “A local search mechanism for peer-to-peer networks,” proceedings of 11th International Conference on Information and Knowledge Management, pages 300–307, NY, USA, 2002.

[23] C. Ng, K. Sia, “Peer Clustering and Firework Query Model,” proceeding of 11th World Wide Web Conference, 2002.

[24] E. Cohen, A. Fiat, H. Kaplan, “Associative Search in Peer-to-Peer Networks: Harnessing Latent Semantics,” proceedings of 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, volume2, pp. 1261–1271, 2003.

[25] M. Bawa, G. Manku, P. Raghavan, “SETS: Search Enhanced by Topic Segmentation,” proceedings of 26th Annual International Conference on Research and Development in Information Retrieval, pages 306–313, NY, USA, 2003.

[26] V. Cholvi, P. Felber, E. Biersack, “Efficient Search in Unstructured Peer-to-Peer Networks,” In Proc. 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures, pages 271–272, NY, USA, ACM Press, 2004.

[27] H. M. Tran, J. Schönwälder, "Heuristic Search using a Feedback Scheme in Unstructured Peer-to-Peer Networks," proceedings of 5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing, Vienna, September, Springer LNCS, 2007.

[28] D. Nau, M. Ghallab, P. Traverso, "Automated Planning: Theory & Practice," Morgan Kaufmann Publishers Inc, 2004.

[29] B. Srivastava, S. Kambhampati, "The Case for Automated Planning in Autonomic Computing," proceedings of 2nd International Conference on Autonomic Computing (ICAC'05), 2005.

[30] B. Srivastava, S. Kambhampati, "Challenges in Adapting Automated Planning for Autonomic Computing," proceedings of 2005 International Conference on Automated Planning and Scheduling (ICAPS'05), 2005.

[31] A. Andrzejak, U. Herman, A. Sahai, "FEEDBACKFLOW - An Adaptive Workflow Generator for Systems Management," proceedings of 2nd International Conference on Autonomic Computing, ICAC, 2005.

[32] M. Ghallab, E. Nationale, C. Aeronautiques, C. K. Isi, S. Penberthy, D. E. Smith, Y. Sun, D. Weld, "PDDL – The Planning Domain Definition Language," proceedings of 4th International Conference on Artificial Intelligence Planning Systems (AIPS'98), 1998.

[33] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, P. Traverso, "MBP: A Model based Planner," proceedings of 2003 International Conference on Automated Planning and Scheduling (ICAPS'03), 2003.

[34] N. Arshad, D. Heimbigner, A. L. Wolf, "Deployment and dynamic reconfiguration planning for distributed software systems," Software Quality Journal, 2007.

[35] A. Gerevini, I. Serina, "LPG: A Planner Based on Local Search for Planning Graphs with Action Costs," proceedings of 6th International Conference on AI Planning and Scheduling, 2002.

[36] Fink, "Modelling and Assisting the Design of IT Changes," Master Thesis, Ludwig-Maximillians-University of Munich, 2008.

[37] Office of Government Commerce (OGC), ed.: Service Support IT Infrastructure Library (ITIL). The Stationery Office, Norwich, UK, 2000.

[38] L. Xu, B. Y. Choueiry, "A new efficient algorithm for solving the simple temporal problem," proceedings of 10th International Symposium on Temporal Representation and Reasoning, 2003.

[39] N. Arshad, D. Heimbigner, A. L. Wolf, "A planning based approach to failure recovery in distributed systems," proceedings of 1st ACM SIGSOFT Workshop on Self-Managed Systems, 2004.

[40] N. Arshad, D. Heimbigner, A. L. Wolf, "Dealing with failures during failure recovery of distributed systems," ACM SIGSOFT Software. Eng. Notes, 2005.

[41] A. Ranganathan, R. H. Campbell, "Autonomic Pervasive Computing Based on Planning," proceedings of 1st International Conference on Autonomic Computing (ICAC'04), 2004.

[42] H. A. Kautz, B. Selman, "Unifying SAT-based and Graph-based Planning," proceedings of 16th International Joint Conference on Artificial intelligence, 1999.

[43] R. E. Fikes, N. J. Nilsson, “STRIPS: a new approach to the application of theorem proving to problem solving,” In Computation & intelligence: Collected Readings American Association for Artificial Intelligence, 1995.

[44] J. Blythe et al., “Transparent Grid Computing: A Knowledge-Based Approach,” In Proc. of the 15th Annual Conf. on Innovative Applications of Artificial Intelligence (IAAI), 2003.

[45] J. Blythe et al., “The Role of Planning in Grid Computing,” In Proc. of 13th International Conf. on Automated Planning and Scheduling, 2003.

[46] T. Buchholz, T. M. Krause, C. Linnhoff-Popien, M. Schiffers, “CoCo: Dynamic Composition of Context Information,” Proceedings of the First Annual International Conference on Mobile and Ubiquitous Computing (MobiQuitous), pp. 335-343, August 2004.

[47] F. Fuchs, I. Hochstatter, M. Krause, M. Berger, “A Meta–Model Approach to Context Information,” proceedings of 3rd IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom 2005), pp. 8-14, March 2005.

[48] Jena Semantic Web Framework, http://jena.sourceforge.net/, 2005.

[49] World Wide Web Consortium: Semantic Web, http://www.w3.org/2001/sw/, 2005.

[50] K. Wilkinson, C. Sayers, H. Kuno, D. Reynolds, “Efficient RDF Storage and Retrieval in Jena2”, proceedings of VLDB Workshop on Semantic Web and Databases, September 2003.

[51] World Wide Web Consortium: SPARQL Query Language for RDF, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/, January 2008.

[52] F. Fuchs, “A Modelling Technique for Context Information,” Master’s Thesis, Ludwig Maximilian University Munich, 2004.

[53] T. Strang, C. Linnhoff-Popien, “A Context Modelling Survey,” proceedings of the Workshop on Advanced Context Modelling, Reasoning and Management, September 2004.

[54] Telecommunications Management Networks - Management Services approach - Enhanced Telecommunications Operations Map (eTOM), http://www.catr.cn/cttlcds/itu/itut/product/bod, March 2007.

[55] Telecommunications Management Networks - Principles for the Management of Next Generation Networks. http://www.catr.cn/cttlcds/itu/itut/product/bodyM.htm, March 2006.

[56] J. M. Serrano, J. Serrat, J. Strassner, M. Ó Foghlú, “Facilitating Autonomic Management for Service Provisioning using Ontology-Based Functions & Semantic Control,” 3rd IEEE International Workshop on Broadband Convergence Networks (BcN 2008) in IEEE/IFIP NOMS 2008, Salvador de Bahia, Brazil, April 2008.

[57] G. Kouadri Mostefaoui, P. Brezillon, “Modelling Context-Based Security Policies with Contextual Graphs,” In Workshop on Context Modelling and Reasoning (CoMoRea’04), March 2004.

[58] J. Park, S. Ram, “Information systems interoperability: What lies beneath?,” ACM Transactions on Information Systems, Volume 22, Issue 4, October 2004.

[59] R. Fileto, C. Bauzer, “A Survey on Information Systems Interoperability,” Technical report – IC-03-030, December 2003.

[60] J. M. Serrano, J. Serrat, J. Strassner, R. Carroll, “Policy-Based Management and Context Modelling Contributions for Supporting Autonomic Systems,” IFIP/TC6 Autonomic Networking. France Télécom, Paris, France, 2006.

[61] T. Gruber, “Toward Principles for the Design of Ontologies Used for Knowledge Sharing,” International Journal of Human-Computer Studies, Volume 43, Issue 5, p.p. 907 – 928, 1995.

[62] J. Strassner, E. Lehtihet, N. Agoulmine, “FOCALE – A Novel Autonomic Computing Architecture,” Latin American Autonomic Computing Symposium (LAACS’06), July 2006.

[63] J. Strassner, D. Raymer, S. Samudrala, G. Cox, Y. Liu, M. Jiang, J. Zhang, S. van der Meer, B. Jennings, M. Ó Foghlú, W. Donnelly, “The Design of a New Context-Aware Policy Model for Autonomic Networking,“ In Proceedings of 5th IEEE International Conference on Autonomic Computing (ICAC 2008) Chicago, Illinois, USA, June 2008.

[64] J. M. Serrano, J. Serrat, J. Strassner, G. Cox, R. Carroll, M. Ó Foghlú, “Services Management Using Context Information Ontologies and the Policy-Based Management Paradigm: Towards the Integrated Management in Autonomic Communications,” First IEEE International Workshop on Autonomic Communications and Network Management (ACNM 2007), in 10th IFIP/IEEE International Symposium on Integrated Management (IM 2007), Munich, Germany, May 2007.

[65] Foundation for Intelligent Physical Agents (FIPA). Quality of Service Ontology Specification, Specification number SC00094, Geneva, Switzerland, 2002.

[66] C. Estan, G. Varghese, “New Directions in Traffic Measurement and Accounting,” ACM SIGCOMM Internet Measurement Workshop, San Francisco, California, U.S.A., pp. 75-80, November 2001.

[67] C. Morariu, B. Stiller, “DiCAP: Distributed Packet Capturing Architecture for High-Speed Network Links,” proceedings of 33rd Annual IEEE Conference on Local Computer Networks (LCN), San-Jose, USA, October 2008.

[68] Bro IDS Homepage: http://www.bro-ids.org/, July 2008.

[69] Snort Homepage: http://www.snort.org/, July 2008.

[70] K. C. Claffy, G. C. Polyzos, H. W. Braun, “Application of Sampling Methodologies to Network Traffic Characterization,” proceedings of ACM SIGCOMM’93, San Francisco, California, U.S.A., pp. 194-203, September 13-17, 1993.

[71] W. Cochran, “Sampling Techniques,” Wiley, 1987.

[72] N. G. Duffield, M. Grossglauser, “Trajectory Sampling for Direct Traffic Observation,” IEEE/ACM Transactions on Networking, Vol. 9, No. 3, pp. 280-292, June 2001.

[73] N. G. Duffield, C. Lund, M. Thorup, “Learn More, Sample Less: Control of Volume and Variance in Network Measurement,” IEEE Transactions of Information Theory, Vol. 51, No. 5, pp. 1756-1775, May 2005.

[74] J. Mai, C. Chuah, A. Sridharan, H. Z. T. Ye, "Is Sampled Data Sufficient for Anomaly Detection?," proceedings of 6th ACM SIGCOMM Conference on Internet Measurement, Rio de Janeiro, Brazil, pp. 165-176, October 25-17, 2006.

[75] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, A. Lakhina, "Impact of Packet Sampling on Anomaly Detection Metrics," proceedings of 6th ACM SIGCOMM Conference on Internet Measurement, Rio de Janeiro, Brazil, pp. 159-164, October 25-17, 2006.

[76] IEEE IPFIX Working Group Homepage: http://www.ietf.org/ html.charters/ipfix-charter.html, April 2008.

[77] Endace Homepage: http://www.endace.com/, March 2008.

[78] libpcap-PFRING Homepage: http://www.ntop.org/PF_RING.html, February 2008.

[79] L. Deri, "High-Speed Dynamic Packet Filtering," Journal of Network and Systems Management, Vol. 15, No. 3, pp. 401-415, September 2007.

[80] Broadcom BCM 5721 Card, http://www.broadcom.com/products/Small-Medium-Business/Gigabit-Ethernet-Controllers/BCM5721, July 2008.

[81] Linux Online Homepage: http://www.linux.org/, March 2008.

[82] C. Morariu, T. Kramis, B. Stiller, "DIPStorage: Distributed Architecture for Storage of IP Flow Records," proceedings of 16th Workshop on Local and Metropolitan Area Networks, Cluj-Napoca, Romania.

[83] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage," proceedings of 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), Cambridge, Massachusetts, USA, pp. 190-201, November 12-15, 2000.

[84] Cisco IOS NetFlow Homepage: http://www.cisco.com/en/US/products/ps6601/ products_ios_protocol_group_home.html, April 2008.

[85] JXTA Homepage: https://jxta.dev.java.net/, April 2008.

[86] FreePastry Homepage: http://www.freepastry.org/, April 2008.

[87] SCRIBE Homepage: http://research.microsoft.com/~antr/scribe/, April 2008.

[88] MySQL Homepage: http://www.mysql.com/, April 2008.

[89] Verteilte Systeme, Universität Konstanz Homepage: http://www.inf.uni-konstanz.de/disy/, April 2008.

[90] XPath Homepage: http:// www.w3.org/TR/xpath/, April 2008.

[91] MulticastDNS Homepage: http://www.multicastdns.org/, April 2008.

[92] JmDNS Homepage: http://streamer.rit.edu/~jeffs/JmDNS/, April 2008.

[93] EU IST Autonomic Internet Project, "D6.1 Autonomic Internet Initial Framework," http://ist-autoi.eu

[94] Future Internet Design (FIND) Program - http://www.nets-find.net/

[95] Global Environment for Network Innovation (GENI) Program - http://www.geni.net/

[96] Ambient Networks (ANs) Project, http://www.ambient-networks.org

[97] Mobile Ad-hoc NETworks (MANETs), http://www.ietf.org/html.charters/manet-charter.html

[98] "AKARI" Architecture Design Project for New Generation Network http://akari-project.nict.go.jp/eng/conceptdesign.htm

[99] S. Wasserman, K. Faust, "Social Networks Analysis: Methods and Applications," Cambridge University Press, 1994.

[100] R. Kumar, P. Ragbavan, S. Rajagopalan, A. Tomkins, "The web and social networks," IEEE Computer, Volume 35, Issue 11, pp. 32-36, November 2002.

[101] H. P. Thadakamalla, U. N. Raghavan, S. Kumara, A. Albert, "Survivability of multiagent-based supply networks: A topological perspective," IEEE Intelligent Systems, 2004.

[102] D. Culler, D. Estrin, M. Srivastava, "Guest editors' introduction: Overview of sensor networks," IEEE Computer, 2004.

[103] B. Horling, V. Lesser, "A Survey of Multi-Agent Organizational Paradigms. The Knowledge Engineering Review," Cambridge University Press, 2005.

[104] C. Merida-Campos, S. Willmott, "Agent compatibility and coalition formation: Investigating two interacting negotiation strategies," proceedings of Agent Mediated Electronic Commerce VIII (TADA/AMEC). LNAI 4452, pp. 75-90, Hakodate, Japan, 2006.

[105] C. Merida-Campos, S. Willmott, "The effect of heterogeneity on coalition formation in iterated request for proposal scenarios," proceedings of the 4th European Workshop on Multi- Agent Systems, Lisbon, 2006.

[106] A. Galis, S. Covaci, "Active Virtual Private Networks Services On Demand," proceedings of the IEEE Interworking 2000 Conference, Bergen, October 2000 and in Springer-Verlag, ISBN 3-540-41140-2, October 2000.

[107] C. Merida-Campos, S. Willmott, "Stable coalitions under different demand conditions in iterative request for proposal environments," International Transactions on Systems Science and Applications, pending of publishing, 2008.

[108] C. Merida-Campos, S. Willmott, "Exploring social networks in request for proposal dynamic coalition formation problems," proceedings of the 5th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS '07), Leipzig, Germany, 2007.

[109] C. Merida-Campos, Steven Willmott, "The impact of betweenness in small world networks on request for proposal coalition formation problems," proceedings of the 10th International Congress of the Catalan Association of Artificial Intelligence, Andorra, Sant Juli`a de L`oria, Andorra, 2007.

[110] Erdős, P.; Rényi, A, "On Random Graphs I," Publicationes Mathematicae 6, pp. 290–297, 1959.

[111] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, D. Noveck, "Network File System (NFS) version 4 Protocol," 2003.

[112] A. Epps, G. Bailey, D. Glatz, "NFS and SMB data sharing within a heterogeneous environment: a real world study," proceedings of the 2nd Conference on Large Installation System Administration of Windows NT, Berkeley, CA, USA, 1999.

[113] J. Dollimore, T. Kindberg, G. Coulouris, "Distributed Systems: Concepts and Design," 4th Edition, International Computer Science Series, Addison Wesley, May 2005.

[114] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, "Looking up data in P2P systems," Communications of the ACM, pp. 43-48, 2003.

[115] A. Rowstron P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pp. 329-350, November 2001.

[116] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, B. Zhao, "OceanStore: an architecture for global-scale persistent storage," SIGARCH Computer Architecture, 28(5):190–201, 2000.

[117] P. Maymounkov, D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), March 2002.

[118] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, R. Campbell, "A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems," proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II, pp. 205-213, Washington, DC, USA, April 2005.

[119] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica, "Wide-area cooperative storage with CFS," SIGOPS Operating Systems Review, pp. 202-215, 2001.

[120] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01), pp. 149-160, New York, NY, USA, 2001.

[121] A. Rowstron, P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," SIGOPS Operating Systems Review, pp. 188-201, 2001.

[122] A. Muthitacharoen, R. Morris, T. M. Gil, B. Chen, "Ivy: a read/write peer-to-peer file system," SIGOPS Operating Systems Review, pp. 31-44, 2002.

[123] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, J. Kubiatowicz, "Pond: The OceanStore Prototype," proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03), pp. 1-14, Berkeley, CA, USA, 2003.

[124] K. Hildrum, J. D. Kubiatowicz, S. Rao, B. Y. Zhao, "Distributed object location in a dynamic network," proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures (SPAA '02), pp. 41-52, New York, NY, USA, 2002.

[125] I. Gupta, K. Birman, P. Linga, A. Demers, R. van Renesse, "Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead," proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), 2003.

[126] B. Amann, B. Elser, Y. Houri, T. Fuhrmann, “IgorFs: A Distributed P2P File System,” proceedings of 8th International Conference on Peer-to-Peer Computing (P2P ’08), pp. 77-78, September 2008.

[127] W. K. Lin, D. M. Chiu, Y. B. Lee, “Erasure code replication revisited,” proceedings of 4th International Conference on Peer-to-Peer Computing, pp. 90-97, 2004.

[128] R. Rodrigues B. Liskov, “High Availability in DHTs: Erasure Coding vs. Replication,” proceedings of 4th International Workshop on Peer-to-Peer Systems (IPTPS ‘05), pp. 226–239, 2005.

[129] TomP2P, a distributed multi map, http://www.csg.uzh.ch/publications/software/TomP2P, December 2008.

[130] Filesystem in Userspace, August 2008. http://fuse.sourceforge.net/, December 2008.

[131] M. J. Bach, “The design of the UNIX operating system,” Prentice Hall International, London, 1986.

[132] R. Bhagwan, D. Moore, S. Savage, G. M. Voelker, “Replication Strategies for Highly Available Peer-to-Peer Storage,” proceedings of 2003 Workshop on Future Directions in Distributed Computing, pages 153-158, 2003.

[133] A. S. Tanenbaum, M. van Steen, “Distributed Systems: Principles and Paradigms,” 2nd Edition, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[134] S. Dayal, “Characterizing HEC Storage Systems at Rest,” Technical Report CMU-PDL-08-109, Carnegie Mellon University Parallel Data Lab, July 2008.

[135] N. Agrawal, W. J. Bolosky, J. R. Douceur, J. R. Lorch, “A Five-Year Study of File-System Metadata,” ACM Transactions on Storage (TOS), Vol. 3, No. 3, pp. 31-45, October 2007.

[136] L. Subramanian, I. Stoica, H. Balakrishnan, R. Katz, “Overqos: Offering Internet QoS Using Overlays,” ACM SIGCOMM CCR, Vol. 33, pp. 11-16, January 2003.

[137] B. D. Vleeschauwer, F. D. Turck, B. Dhoedt, P. Demeester, “On the construction of QoS enabled overlay networks,” Quality of Service in the Emerging Networking Panorama, Springer Berlin, pp. 164-173, September 2004.

[138] Z. Li, P. Mohapatra, “Qron: QoS-aware routing in overlay networks,” IEEE Journal on Selected Areas in Communications, Volume 22, Issue 1, pp. 29-40, January 2004.

[139] E. Adar and B. Huberman, “Free Riding on Gnutella,” Technical Report, Xerox PARC, August 2000.

[140] M. Feldman, K. Lai, I. Stoica, J. Chuang, “Robust incentive techniques for peer-to-peer networks,” proceedings of 5th ACM Conference on Electronic Commerce (EC ’04), pp. 102-111, New York, NY, USA, 2004.

[141] S. Sanghavi, B. Hajek, “A new mechanism for the free-rider problem,” proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems (P2PECON ’05), pp. 122-127, New York, NY, USA, 2005.

[142] M. Feldman, J. Chuang, “Overcoming free-riding behaviour in peer-to-peer systems,” SIGecom Exchanges, Volume 5, Issue 4, pp. 41-50, 2005.

[143] J. Feigenbaum, S. Shenker, “Distributed algorithmic mechanism design: recent results and future directions,” proceedings of 6th International Workshop on Discrete algorithms and methods for mobile computing and communications (DIALM ’02), pp. 1-13, New York, NY, USA, 2002.

[144] V. Vishnumurthy, S. Chandrakumar, E. G. Sirer, “Karma: A secure economic framework for p2p resource sharing,” proceedings of the 2003 ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON ’03), 2003.

[145] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, W. S. Stornetta, “Spawn: A distributed computational economy,” IEEE Transactions on Software Engineering, Vol. 18, No. 2, 103-117, 1992.

[146] B. Chun, Y. Fu, A. Vahdat, „Bootstrapping a distributed computational economy,” proceedings of the 2003 ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON ’03), 2003.

[147] D. A. Turner K. W. Ross, “A lightweight currency paradigm for the P2P resource market” proceedings of 7th International Conference on Electronic Commerce Research (ICECR ’04), June 2004.

[148] D. F. Ferguson, C. Nikolaou, J. Sairamesh, Y. Yemini, “Economic models for allocating resources in computer systems,” Market-based control: a paradigm for distributed resource allocation, World Scientific Publishing Co., pp. 156-183, 1996.

[149] W. Wang, B. Li, “Market-driven bandwidth allocation in selfish overlay networks,” proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM ’05), pp. 2578-2589, 2005.

[150] K. Tamilman, V. Pai, A. Mohr, “Swift: A system with incentives for trading,” proceedings of the 2004 ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON ’04), 2004.

[151] Z. Zhang, S. Chen, M. Yoon, “MARCH: A distributed incentive scheme for peer-to-peer networks,” proceedings of 26th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM ’07), pp. 1091-1099, May 2007.

[152] K. G. Anagnostakis, M. B. Greenwald, “Exchange-based incentive mechanisms for peer-to-peer file sharing,” proceedings of 24th International Conference on Distributed Computing Systems (ICDCS ’04), pp. 524-533, Washington, DC, USA, 2004.

[153] J. R. Douceur, “The sybil attack,” Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS ’01), pp. 251-260, London, UK, 2002.

[154] M. Feldman, C. Papadimitriou, J. Chuang, I. Stoica, “Free-riding and whitewashing in peer-to-peer systems,” proceedings of the ACM SIGCOMM Workshop on Practice and theory of incentives in networked systems (PINS ’04), pp. 228-236, New York, NY, USA, 2004.

[155] J. A. Mirrlees, “The theory of moral hazard and unobservable behaviour: Part I,” Review of Economic Studies, Volume 66, Issue 1, pp. 3-21, January 1999.

[156] T. Cholez, I. Chrisment, O. Festor, “A distributed and adaptive revocation mechanism for p2p networks,” proceedings of 7th International Conference on Networking (ICN ’08), pp. 290-295, Washington, DC, USA, 2008.

[157] Moritz Steiner, Taoufik En Najjary, Ernst W Biersack, "Exploiting KAD: possible uses and misuses," Computer Communications Review, Vol. 37, No. 5, pp. 65-70, October 2007.

[158] T. Cholez, I. Chrisment, O. Festor, "Defending the Sybil attack: a study of protection mechanisms in KAD," Submitted to Networking 2009, the 8th event of the series of International Conferences on Networking, December 2008.

[159] H. M. Tran, J. Schönwälder, "Fault Representation in Case-Based Reasoning," proceedings of 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM '07), pp 50-61, Springer-Verlag, 2007.

[160] H. M. Tran, G. Chulkov, J. Schönwälder, "Crawling Bug Tracker for Semantic Bug Search," proceedings 19th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, pp. 55-66, Springer-Verlag, 2008.

[161] G. Chulkov, "Buglook: a search engine for bug reports," Master Thesis, Jacobs University Bremen, August 2008.

[162] D. Bloom, "Selection Criterion and Implementation of a Trouble Tracking System: What's in a Paradigm?" proceedings of 22nd Annual ACM SIGUCCS Conference on User Services (SIGUCCS '94), pp. 201-203. ACM Press, 1994.

[163] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, R. Harshman, "Indexing by Latent Semantic Analysis," Journal of the Society for Information Science, Vol. 41, No. 6, 391-407, 1990.

[164] M. W. Berry, Z. Drmac, E. R. Jessup. Matrices, "Vector Spaces, and Information Retrieval," Society for Industrial and Applied Maths Review, Vol. 41, No. 2, pp. 335-362, 1999.

[165] P. Haase, J. Broekstra, M. Ehrig, M. Menken, P. Mika, M. Plechawski, P. Pyszlak, B. Schnizler, R. Siebes, S. Staab, C. Tempich," Bibster - a semantics-based bibliographic peer-to-peer system," proceedings of 3rd International Semantic Web Conference, pp. 122-136. Springer-Verlag, 2004.

[166] R. R. Yager, "On ordered weighted averaging aggregation operators in multi-criteria decision making," IEEE Transactions on Systems, Man, and Cybernetics, Volume 18, Issue 1, pp. 183-190, 1988.

[167] Python programming language – Official Website. http://www.python.org/, 20 August 2008.

[168] Django - the web framework for perfectionists with deadlines. http://www.djangoproject.com/, 20 August 2008.

[169] Python package index: shove. http://pypi.python.org/pypi/shove/, 20 August 2008.

[170] Twisted. http://twistedmatrix.com/trac/, 20 August 2008.

[171] SVDLIBC. http://tedlab.mit.edu/dr/svdlibc, retrieved on 20 August 2008.

10 Abbreviations

AC	Admission Control
AI	Artificial Intelligence
AMS	Autonomic Management System
AOO	Autonomic Orchestration Overlay
AP	Automated Planning
AVP	Attribute Value-Pair
BTS	Bug Tracking System
BW	Bandwidth
CAS	Context Aware Services
CBR	Case-Based Reasoning
CFS	Cooperative File System
CIS	Context Information Services
CMDB	Change Management Database
CMM	Context Meta-Model
CoCo	Composing Context
CPU	Central Processing Unit
DHT	Distributed Hash Table
DiffServ	Differentiated Services
DMM	Distributed Multi-Map
DNS	Domain Name System
DOC	Distributed Orchestration Component
DOM	Document Object Model
DRAM	Dynamic Random Access Memory
DRsM	Dynamic Resource Management
DRFS	Distributed Reliable File System
EC	Event Calculus
FB Search	Feedback Search Mechanism
FD Search	Flooding Search Mechanism
FIN	Future Internet Network
FTP	File Transfer Protocol
FUSE	Filesystem in Userspace
HTML	HyperText Markup Language
HTN	Hierarchical Task Network

IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPFIX	Internet Protocol Flow Information eXport
ISP	Internet Service Provider
IT	Information Technology
JRE	Java Runtime Environment
KD	Knowledge Discovery
LAN	Local Area Network
LPG	Local search for Planning Graphs
LPP	Logic Programming Priorities
LPwNF	Logic Programming without Negotiation as Failure
LRU	Least Recently Used
LSI	Latent Semantic Index
LSP	Labelled Switch Path
MBP	Model Based Planner
MPLS	Multi-Protocol Label Switching
ND	Network Dimensioning
NFS	Network File System
NGN	Next Generation Network
NIC	Network Interface Card
OQL	Overall Quality Level
OSM	Ontology for Support and Management
OWL	Web Ontology Language
P2P	Peer-to-Peer
PDDL	Planning Domain Definition Language
PMA	Policy Management Agent
PMT	Policy Management Tool
POJO	Plain Old Java Object
QC	Quality of Service Class
QoS	Quality of Service
RAB	Resource Availability Buffer
RD Search	Random Search Mechanism
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema

RPC	Remote Procedure Call
RSS	Really Simple Syndication
SCTP	Stream Control Transmission Protocol
SLS	Service Level Specification
SLS-S	Service Level Specification-Subscription
SLS-I	Service Level Specification-Invocation
SMB	Server Message Block
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SPOF	Single Point of Failure
SQL	Structured Query Language
STP	Simple Temporal Problem
SVD	Singular Value Decomposition
SU	Subscription Upper
TCL	Traffic Critical Level
TCP	Transmission Control Protocol
TE	Traffic Engineering
TT	Traffic Trunk
TTS	Trouble Ticket System
UDP	User Datagram Protocol
VCL	Very Critical Level
VE	Virtual Enterprise
VSM	Vector Space Model
VSP	Virtual Service Provider
XML	eXtensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

11 Acknowledgements

This deliverable was made possible due to the large and open help of the WP9 team of the EMANICS consortium within the Network of Excellence, which includes all of the deliverable authors as indicated in the document control. We would like to thank all of them.